

DTIC FILE COPY

E 901781

(2)

AFATL-TR-88-117, VOL III

Program EAGLE User's Manual

Vol III-Grid Generation Code

AD-A204 143

Joe F Thompson
Boyd Gatlin

DEPARTMENT OF AEROSPACE ENGINEERING
MISSISSIPPI STATE UNIVERSITY
DRAWER A
MISSISSIPPI STATE, MS 39762

SEPTEMBER 1988



INTERIM REPORT FOR PERIOD OCTOBER 1986 - SEPTEMBER 1988

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AIR FORCE ARMAMENT LABORATORY

Air Force Systems Command ■ United States Air Force ■ Eglin Air Force Base, Florida

88 1011 224

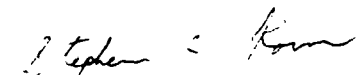
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



STEPHEN C. KORN
Technical Director, Aeromechanics Division

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization, please notify AFATL/FXA, Eglin AFB FL 32542-5434.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S) AFATL-TR-88-117, Vol III		
6a. NAME OF PERFORMING ORGANIZATION Dept. of Aerospace Eng. Mississippi State University		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Aerodynamics Branch Aeromechanics Division		
6c. ADDRESS (City, State, and ZIP Code) Drawer A Mississippi State, MS 39762			7b. ADDRESS (City, State, and ZIP Code) Air Force Armament Laboratory Eglin Air Force Base, FL 32542-5434		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Aeromechanics Division		8b. OFFICE SYMBOL (If applicable) AFATL/FXA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F08635-84-C-0228		
8c. ADDRESS (City, State, and ZIP Code) Air Force Armament Laboratory Eglin Air Force Base, FL 32542-5434			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO. 62602F	PROJECT NO. 2567	TASK NO. 03
11. TITLE (Include Security Classification) Program EAGLE User's Manual, Volume III: Grid Generation Code					
12. PERSONAL AUTHOR(S) Joe F. Thompson, Boyd Gatlin					
13a. TYPE OF REPORT Interim		13b. TIME COVERED FROM Oct 86 TO Sep 88		14. DATE OF REPORT (Year, Month, Day) September 1988	
15. PAGE COUNT 390					
16. SUPPLEMENTARY NOTATION Availability of this report is specified on verso of front cover. This Volume is a joint AFATL and Contractor effort. Therefore it is in contractor format.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Numerical Grid Generation Algebraic Grid Generation Elliptic Grid Generation		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This report provides both a user's manual and a detailed description of operation for a general three-dimensional grid generation code for the construction of composite grids in arbitrary regions. The code can operate either as an algebraic generation system or as an elliptic generation system. Provision is made for orthogonality at boundaries and complete continuity at block interfaces. The code can operate in two or three dimensions, or on a curved surface. The input is structured to be user-oriented, and arbitrary block configurations can be treated.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL John R. Cipolla			22b. TELEPHONE (Include Area Code) (904) 882-3124		22c. OFFICE SYMBOL AFATL/FXA

SUMMARY

This report provides both a user's manual and a detailed description of operation for a general three-dimensional grid generation system for the construction of composite grids in arbitrary regions. The code can operate either as an algebraic generation system or as an elliptic generation system. Provision is made for orthogonality at boundaries and complete continuity at block interfaces. The system can operate in two or three dimensions, or on a curved surface. The input is structured to be user-oriented, and arbitrary block configurations can be treated.



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
Availability	
Availability	
Dist	Special
A-1	

PREFACE

Volume III of a four-volume set documents the usage of Program EAGLE (Eglin Arbitrary Geometry Implicit Euler) to generate a discrete set of points or cells covering the physical field. The Grid program creates the physical field required for the finite difference (finite volume) or finite element solution in the computational domain. The Grid program can serve as input to the EAGLE Flow-Solver (Volume IV) or some other numerical analysis program using the finite element or finite difference technique.

This report was prepared by Drs Joe F. Thompson and Boyd Gatlin of Mississippi State University (MSU), Starkville, MS. The work was performed under Work Unit 25670308 from 1 October 1986 to 30 September 1988.

The principal investigator of the surface and grid generation theory has been Dr Joe F. Thompson of MSU. The principal investigators of the flow code theory have been Dr David L. Whitfield of MSU, and Dr Dave M. Belk and Mr L. Bruce Simpson of the Computational Fluid Dynamics Section (AFATL/FXA). Capt Jon S. Mounts of the Computational Fluid Dynamics Section (AFATL/FXA) has increased the utility of the flow code through user-oriented inputs and outputs, extensive error checking, and calculation of component forces and moments.

The program manager for the development of Program EAGLE has been Dr Lawrence E. Lijewski of the Computational Fluid Dynamics Section.

ACKNOWLEDGEMENT

This project has involved cooperative efforts between the contractor and the sponsor, and Dr Lawrence Lijewski of Eglin Air Force Base has contributed significantly both through direction and involvement. Codes of this size evolve as usage uncovers problems and new requirements. In this regard, Lt Augusto Martinez has been instrumental, and was also heavily involved in the incorporation of the use of the solid state disk. Dr Dave Belk, Mr L. Bruce Simpson, Mr Yen Tu, Mr John Cipolla, Mr Lynn Lewis, Capt Jon Mounts, and Lt Montgomery Hughson of Eglin have also contributed. Graduate students Yeon Seok Chae, Col Hyun Jun Kim, and Maj Yong Hyun Yoon of Mississippi State University have contributed significantly to the checkout of these codes. Mr Chae did the artwork. Finally, Mrs Susan Triplett Price of Mississippi State did all the typing with competence, dedication, and extra hours.

This effort was conducted during the period 1 October 1986 to 30 September 1988 for the Air Force Armament Laboratory, Eglin Air Force Base, FL 32542-5434, under Contract F08635-84-C-0228.

TABLE OF CONTENTS

PART I - USER'S MANUAL

A. INTRODUCTION	15
1. Composite Grid Structure	15
2. Block Interfaces	18
3. Fundamental Arrays	20
4. Scratch Files.	23
5. Adjustable Dimension Parameters.	23
B. SET-UP PROCEDURE	25
1. Block Structure.	25
2. Boundary Points.	27
3. Interfaces	28
4. Grid	29
5. Input.	29
C. NAMELIST/INPUT/.	31
1. Calculation and Storage of Input Parameters for Later Use. .	32
2. Block Specifications	34
3. Reading of Boundary Points	36
4. Fixed Points	41
5. Out Points	43
6. Section Interpolation.	43
7. Interface Linkage.	51
8. Reflective Boundary.	55
9. Boundary Orthogonality through Neumann Boundary Conditions .	56
10. Boundary Orthogonality through Control Functions	58
11. Off-Boundary Spacing	60
12. Special Points	60
13. Smoothing the Grid or Control Functions.	62
14. Algebraic Grid	63
15. Control Functions.	64
16. Sub-Block Structure.	67
17. Storage.	68
18. Iterative Solution Parameters.	69
19. Jacobian Check	70
20. Derivative Difference Forms.	70
21. Block Storage.	72
22. Restart.	72
23. Numbered Points.	73
24. 2D Grid on Curved Surface.	78
25. Continuity Check	80
26. Zero-Curvature Extrapolation at Boundary	81

D. NAMELIST/OUTPUT/	82
1. Printout	82
2. Plot File.	83
3. Error Norm	83
E. ERROR MESSAGES	84

PART II - CODE OPERATION

A. INTRODUCTION	103
1. Basic Notation	103
2. Adjustable Dimension Parameters.	103
3. Field Arrays	105
4. Index Functions.	109
5. Block Storage.	110
6. Kroneker Delta	111
7. Cyclic Array	112
8. Loop Order	113
9. Coordinate Derivatives	113
10. Metric Elements.	115
B. INPUT PHASE.	116
1. ITEM="BLOCK"	116
2. ITEM="SUB"	118
3. ITEM="FILE"	118
4. ITEM="LIST"	119
5. ITEM="FIX"	119
6. ITEM="OUT"	119
7. ITEM="NEUMANN"	119
8. ITEM="NEUMAN1" or "NEUMAN2" or "NEUMAN3"	119
9. ITEM="ORTHOG"	120
10. ITEM="REFLECT"	120
11. ITEM="CUT"	120
12. ITEM="FIELD"	121
13. ITEM="IMAGE"	121
14. ITEM="AVERAGE"	121
15. ITEM="INTERP"	121
16. ITEM="SMOOTH"	122
17. ITEM="CONTROL"	123
18. ITEM="INITIAL"	123
19. ITEM="SPACE"	123
20. ITEM="SETVAL"	124
21. ITEM="STORE"	124
22. ITEM="SIZE"	124
23. ITEM="UNFIX"	124
24. ITEM="EXTRAP"	125
25. ITEM="POINT"	125
26. ITEM="SEGMENT"	125
27. ITEM="SURFACE"	125

C. SETUP PHASE.	126
1. Derivative Forms	126
2. Sub-Blocks	126
3. Restart.	126
4. Image Points	126
5. Boundary Array for Neumann Boundaries.	127
6. Boundary Array for Orthogonal Boundaries	128
7. Interpolation for Initial Algebraic Grid	128
8. Normals on Reflective Boundaries	130
9. Values at Image Points	131
10. Smooth Coordinates	131
11. Jacobian Check	131
12. Coordinate Extremes.	132
13. Radius of Curvature Scale.	132
14. Control Functions.	133
15. Control Function Extremes.	138
16. Acceleration Parameters.	138
17. Orthogonal Boundary Initialization	138
18. Default of Control Functions	138
19. Curved Surface Spline.	139
D. OUTPUT PHASE	140
1. ITEM="INITIAL"	140
2. ITEM="PRINT"	141
3. ITEM="PLOT".	141
4. ITEM="ERROR"	141
E. GENERATION PHASE	142
1. Iteration.	142
2. Output	143
F. SUBROUTINES.	144
1. Classes.	144
2. SUBROUTINE SSD	145
3. SUBROUTINE IMGIMG.	146
Special Points	146
Ambiguities.	147
Point Classification Check and Sort.	150
4. SUBROUTINE FIXIMG.	150
5. SUBROUTINE IMGPTS.	151
6. SUBROUTINE SETIMR.	152
Average Points	152
Image Points	153
Other Points	154

7.	SUBROUTINE SETIMP.	154
	Average Points	155
	Image Points	155
	Reflective Points.	157
	Other Boundary Points.	157
8.	SUBROUTINE TRANS	158
	Arrays	158
	Setup.	162
	Arc Length	164
	Index Array.	168
	Boundary Array	170
	Interpolation Fractions.	175
	Blending Functions	177
	Interpolation.	178
9.	SUBROUTINE TERP1	182
10.	SUBROUTINE TERP2	184
11.	SUBROUTINE TERP3	186
12.	SUBROUTINE TERPR	187
13.	SUBROUTINE CONSURF	187
14.	SUBROUTINE OFF	196
15.	SUBROUTINE CONLINE	200
16.	SUBROUTINE CONSURR	204
17.	SUBROUTINE CONLINR	208
18.	SUBROUTINE CONINT.	211
19.	SUBROUTINE VOLSYS.	212
20.	SUBROUTINE OPTACC.	219
21.	SUBROUTINE SMOOTH.	223
22.	SUBROUTINE CONSETU	224
23.	SUBROUTINE CONSETE	226
24.	SUBROUTINE SPLSUR.	231
25.	SUBROUTINE SPLCUR.	232
26.	SUBROUTINE DEFAULT.	233
27.	SUBROUTINE SETYPE.	234
28.	SUBROUTINE SETRES.	234
29.	SUBROUTINE SETNEU.	234
30.	SUBROUTINE SETORT.	235
31.	SUBROUTINE SETREF.	236
32.	SUBROUTINE SETOBJ.	237
33.	SUBROUTINE SETSPA.	237
34.	SUBROUTINE SETSPL.	238
35.	SUBROUTINE SETAXS.	240
36.	SUBROUTINE SETR2D.	240
37.	SUBROUTINE SETNOR.	241
38.	SUBROUTINE SETACC.	242
39.	SUBROUTINE SETIMG.	242
40.	SUBROUTINE SETIMO and SETIMI	244
41.	SUBROUTINE NEUIDX.	249
42.	SUBROUTINE ORIDX	249
43.	SUBROUTINE JACBCK.	250
44.	SUBROUTINE EXTCOR.	250

45.	SUBROUTINE CONFUN.	250
46.	SUBROUTINE EXTCON.	251
47.	SUBROUTINE NEUPTS.	251
48.	SUBROUTINE REFPTS.	261
49.	SUBROUTINE R2DPTS.	262
50.	SUBROUTINE AVGCON.	263
51.	SUBROUTINE REFCON.	263
52.	SUBROUTINE BONCON.	263
53.	SUBROUTINE DEFCON.	264
54.	SUBROUTINE CUTCON.	264
55.	SUBROUTINE CHKTYP.	264
56.	SUBROUTINE CHKSUB.	265
57.	SUBROUTINE CHKINT.	265
58.	SUBROUTINE CHKSET.	265
59.	SUBROUTINE LIM.	265
60.	SUBROUTINE SSDW.	267
61.	SUBROUTINE SSDR.	268
62.	SUBROUTINE SSD1.	268
63.	SUBROUTINE SSD3.	268
64.	SUBROUTINE CCDW.	268
65.	SUBROUTINE CCDR.	269
66.	SUBROUTINE CCD1.	269
67.	SUBROUTINE CCD2.	269
68.	SUBROUTINE CCD3.	269
69.	SUBROUTINE READF.	269
70.	SUBROUTINE READL.	270
71.	SUBROUTINE PRGGRD.	270
72.	SUBROUTINE WRTPLT.	271
73.	SUBROUTINE WRTXYZ.	271
74.	SUBROUTINE WRTGRD.	271
75.	SUBROUTINE WRTRRR.	272
76.	SUBROUTINE WRTRES.	272
77.	SUBROUTINE REDRES.	272
78.	SUBROUTINE AITKEN.	272
79.	SUBROUTINE SING, COSG, TANG, ACOSG	273
80.	SUBROUTINE CHKNEU.	273
81.	SUBROUTINE CHKORT.	273
82.	SUBROUTINE CHKREF.	274
83.	SUBROUTINE CHKCON.	274
84.	SUBROUTINE READB.	274
85.	SUBROUTINE READS.	274
86.	SUBROUTINE BLKCON.	275
87.	SUBROUTINE MOVEIN.	275
88.	SUBROUTINE SURSYS.	275
89.	SUBROUTINE SPLINT.	278
90.	SUBROUTINE SURCOR.	280
91.	SUBROUTINE MERP2, MERP3.	280
92.	SUBROUTINE DERP3.	280

APPENDIX A

TRANSFINITE INTERPOLATION.	281
Projectors.	284
Coding.	293

APPENDIX B

ELLIPTIC GENERATION SYSTEM	300
Laplace system.	300
Poisson system.	301
Effect of boundary point distribution	303
General Poisson-type system	305
Generation system	308

APPENDIX C

CONTROL FUNCTIONS.	309
General development	309
Evaluation along a coordinate line.	318
Evaluation on a coordinate surface.	325

APPENDIX D

ITERATIVE ADJUSTMENT OF CONTROL FUNCTIONS FOR BOUNDARY ORTHOGONALITY.	329
General development	330
Implementation.	333

APPENDIX E

SPLINE	337
------------------	-----

APPENDIX F

NORMAL ON SPLINED SURFACE.	339
------------------------------------	-----

APPENDIX G

INTERPOLATION FOR RADIUS OF CURVATURE.	345
--	-----

APPENDIX H

FEEDBACK LIMITATION ON ACCELERATION PARAMETERS	353
--	-----

APPENDIX I

PREVENTION OF ONE-DIMENSIONAL OVERLAP.	355
--	-----

APPENDIX J

ZERO-CURVATURE INTERSECTION ON SPLINED SURFACE 356

APPENDIX K

SURFACE GRID GENERATION SYSTEMS. 358
OPERATIONS LIST - INPUT. 370
OPERATIONS LIST - OUTPUT 371
SUBROUTINES LIST 372
REFERENCES 375

INTRODUCTION

Finite difference (or finite volume) and finite element solutions both require a discrete set of points or cells covering the physical field, and the efficiency of the computation is greatly enhanced if there is some organization to this set. This organization can be provided by having the discretization defined by the nodes of a curvilinear coordinate system filling the physical field. Such systems are readily available from handbooks for certain simple configurations such as regions that are cylindrical, spherical, elliptical, etc. For general regions of arbitrary shape, numerical grid generation provides the curvilinear system.

The techniques of numerical grid generation, and its application to the numerical solution of partial differential equations, are covered in detail in a recent text on the subject¹. Several surveys of the field have also been given²⁻⁵, and four conference proceedings dedicated to the area have appeared⁶⁻⁹. The first of these proceedings also contains a number of expository papers and other sources on the subject.

The curvilinear system can be constructed simply by setting values in a rectangular array of position vectors:

$$r_{ijk} \quad (i=1,2,\dots,I; \quad j=1,2,\dots,J, \quad k=1,2,\dots,K)$$

and identifying the indices i,j,k with the three curvilinear coordinates. The position vector r is a three-vector giving the values of the x,y , and z Cartesian coordinates of a grid point. Since all increments in the curvilinear coordinates cancel out of the transformation rela-

tions for derivative operators, there is no loss of generality in defining the discretization to be on integer values of these coordinates.

Fundamental to this curvilinear coordinate system is the coincidence of some coordinate surface with each segment of the boundary of the physical region, in the same manner that surfaces of constant radius coincide with the inner and outer boundary segments of the region between two concentric spheres filled with a polar coordinate system. This is accomplished by placing a two-dimensional array of points on a physical boundary segment and setting these values in the array of position vectors with one index constant, e.g., in r_{ijk} with i from 1 to I and j from 1 to J . The curvilinear coordinate k is thus constant on this physical boundary segment. With values set on the sides of the rectangular array of position vectors in this manner, the generation of the grid is accomplished by determining the values of r_{ijk} in the interior of the rectangular array from the specified boundary values on its sides, e.g., by interpolation or a partial differential equation solution. The set of values r_{ijk} then forms the nodes of a curvilinear coordinate system filling the physical region. A physical region bounded by six generally curved sides can thus be considered to have been transformed to a rectangular computational region on which the curvilinear coordinates (i.e., the indices i, j, k) are the independent variables.

Although in principle it is possible to establish a correspondence between any physical region and a single empty rectangular block for general three-dimensional configurations, the resulting grid is likely to be much too skewed and irregular to be usable when the boundary

geometry is complicated. A better approach with complicated physical boundaries is to segment the physical region into contiguous subregions, each bounded by six curved sides (four in 2D) and each of which transforms to a rectangular block in the computational region, with a grid generated within each sub-region¹⁰⁻¹¹. Each sub-region has its own curvilinear coordinate system irrespective of that in the adjacent sub-regions.

This then allows both the grid generation and numerical solutions on the grid to be constructed to operate in a rectangular computational region, regardless of the shape or complexity of the full physical region. The full region is treated by performing the solution operation in all of the rectangular computational blocks. With the composite framework, partial differential equation solution procedures written to operate on rectangular regions can be incorporated into a code for general configurations in a straightforward manner, since the code only needs to treat a rectangular block. The entire physical field then can be treated in a loop over all the blocks.

The generally curved surfaces bounding the sub-regions in the physical region form internal interfaces across which information must be transferred, i.e., from the sides of one rectangular computational block to those of another. These interfaces occur in pairs, an interface on one block being paired with another on the same, or another, block, since both correspond to the same physical surface. Grid lines at the interfaces may meet with complete continuity, with or without slope continuity, or may not even meet.

Complete continuity of grid lines across the interface requires that the interface be treated as a branch cut on which the generation system is solved just as it is in the interior of blocks. The interface locations are then not fixed, but are determined by the grid generation system. This is most easily handled in coding by providing an extra layer of points surrounding each block. Here the grid points on an interface of one block are coincident in physical space with those on another interface of the same or another block, and also the grid points on the surrounding layer outside the first interface are coincident with those just inside the first interface, and vice versa. This coincidence can be maintained during the course of an iterative solution of an elliptic generation system by setting the values on the surrounding layers equal to those at the corresponding interior points after each iteration. All the blocks are thus iterated to convergence together, so that the entire composite grid is generated at once.

The construction of flow codes for complicated regions is greatly simplified by the composite grid structure since, with the use of the surrounding layer of points on each block, a flow code is only required basically to operate on a rectangular computational region. The necessary correspondence of points on the surrounding layers (image points) with interior points (object points) is set up by the grid code and made available to the computational fluid dynamics solution code.

The present grid code is a general three-dimensional elliptic grid generation system based on the block structure. This system allows any number of blocks to be used to cover an arbitrary three-dimensional region. Any block can be linked to any other block (or to itself), with

complete (or lesser) continuity across the block interfaces as specified by input. This code uses an elliptic generation system with automatic evaluation of control functions from the boundary point distributions. This evaluation differs from earlier related procedures¹¹ in that the arc length and curvature contributions to the control functions are evaluated and interpolated separately into the field from the appropriate boundaries. The control function at each point in the field is then formed by combining the interpolated components. This procedure allows very general regions, with widely varying boundary curvature, to be treated.

The control functions can be also determined automatically to provide orthogonality at boundaries with specified normal spacing (related to the GRAPE code¹²). In the present code, the iterative adjustments in the control functions are made by increments radiated from boundary points where orthogonality has not yet been attained. This allows the basic control function structure evaluated from the boundary point distributions to be retained and thus relieves the iterative process from the need to establish this basic form of the control functions.

Alternatively, boundary orthogonality can be achieved through Neumann boundary conditions which allow the boundary points to move over a surface spline, the boundary point locations being located by Newton iteration on the spline to be at the foot of normals to the adjacent field points. Provision is also made for mirror-image reflective boundary conditions, i.e., symmetry planes.

The elliptic generation system is solved by point SOR iteration using a field of locally-optimum acceleration parameters¹³. These optimum parameters make the solution robust and capable of convergence with strong control functions. The code includes an algebraic three-dimensional generation system based on transfinite interpolation¹⁴⁻¹⁵ (using either Lagrange or Hermite interpolation) for the generation of an initial solution to start the iterative solution of the elliptic generation system. This feature also allows the code to be run as an algebraic generation system if desired. The interpolation, though defaulted to complete transfinite interpolation from all boundaries, can be restricted by input to any combination of directions or lesser degrees of interpolation, and the form (Lagrange, Hermite, or incomplete Hermite) can be different in different directions.

The composite structure is such that completely general configurations may be treated, the arrangement of the sub-regions being specified by input, without modification of the code. The input is user-oriented and designed for brevity and easy recognition. For example, the establishment of correspondence, i.e., a branch cut, between two blocks requires only the simple input statement

```
$INPUT ITEM = "CUT", START = __, __, __, END = __, __, __,
```

```
BLOCK = __ ISTART = __, __, __, IEND = __, __, __, IBLOCK = __ $
```

where START and E the three indices of two opposite corners of the cut section on one block (BLOCK), while ISTART and IEND give the corners of the corresponding section on the other block (IBLOCK). The code sets up the point correspondence on the surrounding layers for complete continuity without additional input instructions.

The code is written in modular form so that components can be readily replaced, and an adaptive version of the elliptic generation subroutine has also been written. The code is vectorized (CRAY X-MP) wherever practical and includes provision for separate storage of each block on the solid-state disk (or conventional disk) to allow very large grids to be generated.

In addition, an auxiliary front-end code (surface generation system) has also been written to set up boundary data for input to the grid generation system. This auxiliary code, which is discussed in detail in Vol. II, builds boundary segments in response to a series of input commands which again are designed to be user-oriented, brief, and easily recognized. The following features are included:

- (1) generation of generic plane conic-section or cubic curves.
- (2) generation of cubic space curves.
- (3) generation of generic conic-section surfaces.
- (4) generation of cubic surfaces.
- (5) generation of surfaces by stacking, rotating, or blending curves.
- (6) extraction and concatenation of surface segments.
- (7) transformation of surfaces by translation, rotation, and scaling.
- (8) reversal or switching of point progressions on surface.
- (9) establishment of point distributions by curvature and with specified end, or interior, spacings.
- (10) establishment of surface parametric grids by transfinite interpolation.
- (11) generation of tensor-product surfaces(Coon's Patches).
- (12) generation of surfaces by transfinite interpolation.

(13) generation of grids on curved surfaces.

(14) surface intersections.

The use of the grid generation system, including input instructions, is discussed in Part I of this present volume, and the code operation and background theory are discussed in Part II and the Appendices.

NEW FEATURES

The 1988 version of the EAGLE grid code contains a number of new features and improvements of certain features of the original code. The major thrust of these new features is to simplify the construction of complicated boundary configurations. Reference may be made to a new section entitled Easy EAGLE (Section 4 of Volume I) in which the general use of the code is discussed. This new section is intended to provide the user with essential information on the most commonly used features of the code. This section should hopefully allow the new user to quickly get into operation.

The construction of the block structure for complicated regions has been greatly simplified by making provision for the addressing of points by point numbers instead of by the three coordinate indices (Section I-C23). Boundary segments can also be addressed by numbers. These point and segment numbers can be carried from the front-end boundary code into the grid code, and this allows the number of points on the various boundary segments to be changed in the input to the boundary code without requiring changes in the input to the grid code. This feature also makes it much easier to construct the block structure from sketches of the configuration.

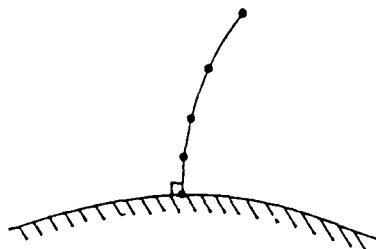
This current version of the code can also generate a 2D grid on a curved surface, including all the features of the composite-block structure, boundary orthogonality, etc. (Section I-C24).

Several bugs in the original version have also been fixed, of course.

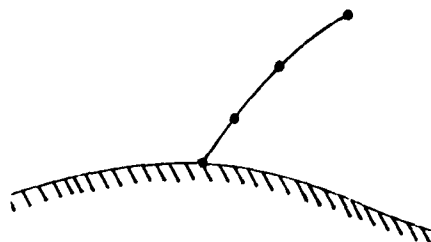
The following significant changes to specific operations should be noted (Details are given in the sections indicated):

Section I-A3

The Neumann boundary conditions may either be orthogonality



or zero curvature.



Section I-B5

With the numbered-points mode (Section I-C23), single entries for START and END, and also for ISTART and IEND, will be taken as point numbers on all input statements involving these quantities.

Section I-C1

Three additional options for MATH are now included.

Section I-C2

When the numbered-points feature (Section I-C23) is used, the block size can be set from a numbered point.

Section I-C3

The default for TRIAD has been changed to "YES".

Since the entries of ORDER set the order of a nest of loops that read the points from the file (with the first entry indicating the innermost loop), it is only necessary to include ORDER when the two directions on the segment of the block are not 1,2, or 2,3, or 1,3, or if the faster direction is higher than the slower direction, e.g. 2,1. In any case, giving the faster running direction as a single entry for ORDER will suffice.

In the numbered point-numbered segment mode (Section I-C23) an entire file of boundary segments is read in at once by including ALL="YES" on an ITEM="FILE" statement instead of BLOCK,START,END,ORDER, and RORDER.

Section I-C4

Points can be "unfixed" by an input statement with ITEM="UNFIX".

Section I-C6

If insufficient boundary values are given for section interpolation, sub-blocks or for the interpolation for the algebraic grid, the code will attempt to supply the missing values by transfinite interpolation of lesser dimensionality.

Section I-C7

Fixed edges can now be included in the definition of cut sections.

Sections I-C8, C9, C10

The section definition for these features can include the fixed edges.

Section I-C9, C10

The off-boundary spacing can be set on the entire section by including SPAVAL, or on portions by the operation ITEM="SPACE", as in Section I-C9.

Section I-C10

The orthogonality section does not have to be on a block side. The extent of the orthogonality into the field can also be controlled.

Section I-C14

It is only necessary to omit ITMAX in order to cause the code to take the algebraic grid as the final result.

Section I-C15

The smoothing of control functions with CONTYP="INITIAL" is automatic. CONTYP="NONE" gives a grid from Laplace equations, i.e., the smoothest possible grid.

Section I-C16

If no points are set on a sub-block boundary, values will automatically be interpolated on that boundary if possible as described in Section I-C6.

Section I-C20

The "VARIABLE" form for the first derivatives now gives central differences with control functions that do not exceed 2.0 in magnitude, with a weighted average between central and one-sided otherwise. The average changes inversely with the control function to fully one-sided for very large values. The default is "VARIABLE".

Section I-C23

It is possible to reference a point by a single number instead of giving the three indices (ξ^1, ξ^2, ξ^3) of the point. This feature, and the companion feature in the front-end boundary code, allows the number of points on a segment to be changed where the segment is generated in the front-end without requiring changes in the input to the grid code.

Section I-C24

The code can generate a 2D grid on a curved surface.

Section I-C25

It is possible in rare instances for cuts to be set up in such a way that a point on a block interface is imaged to a field point in one block, while an adjacent point on the interface is imaged to a non-field

point in another block. Such a situation can produce a local loss of derivative continuity and can often be avoided by reversing the designation of the object and image blocks at the interface. This situation can be checked for by including CONTIN="YES" on some input statement.

Section I-C9

Neumann boundary conditions based on zero curvature of lines intersecting the boundary instead of orthogonality can also be applied.

Section I-D1, D2

As in the NAMELIST/INPUT/, numbered points can be used as single entries in START and END (cf. Section I-C23).

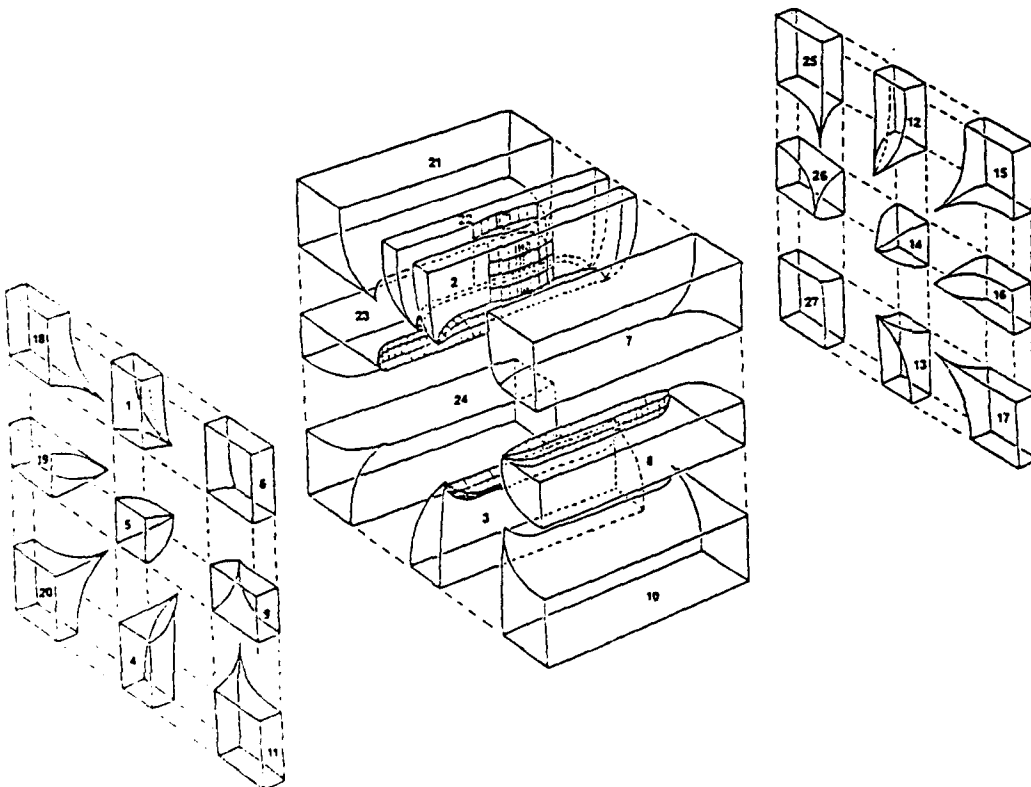
PART I - USER'S MANUAL

A. INTRODUCTION

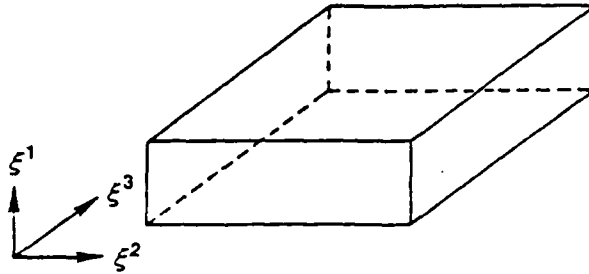
All of the input for the code is discussed in Part I. However, it may be necessary to adjust certain array dimensions for larger applications. The code checks these dimensions and gives error messages and instructions for the necessary adjustments. A list of all such adjustable dimensions is given in Section II-A2 of Part II. Note that any such changes must be done in global edits.

1. Composite Grid Structure

The grid is structured as follows: The entire three-dimensional physical region is filled with a set of interfacing hexahedrons, each of which corresponds to a rectangular computational block.



Each of these computational blocks has its own set of right-handed curvilinear coordinates, $\xi^i (i=1,2,3)$, independent of those in the other blocks:



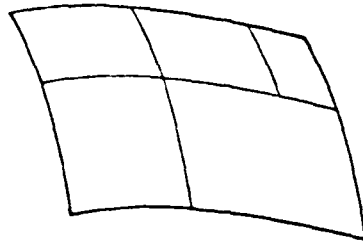
Each block is identified by a number (starting with 1), and the size (the number of grid points in each curvilinear direction) of a block is set in the integer array

$$\text{CMAX}(i, \text{block number}) \quad i = 1,2,3$$

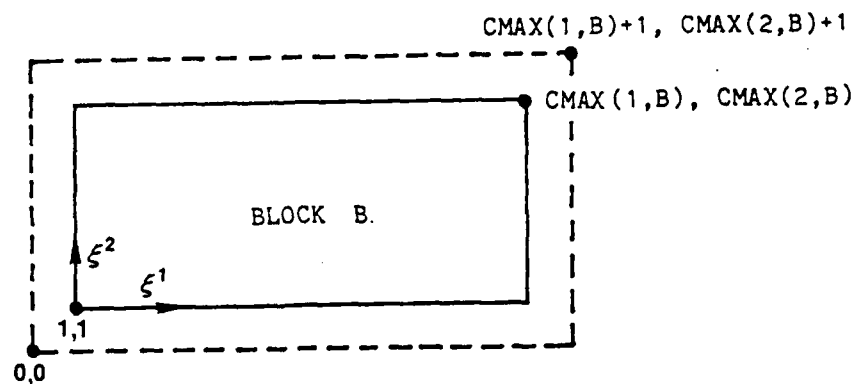
The curvilinear coordinates of the grid points in the block thus assume the integer values

$$\xi^i = 1,2,\dots, \text{CMAX}(i, \text{block number}) \quad i = 1,2,3$$

at the grid points in this computational block. The blocks do not have to be all of the same size, and the size of each is specified by input. It is only necessary that all of the corresponding hexahedrons fit together to fill the physical region.



Each computational block is surrounded by an extra layer of points in order to allow connections across the interfaces in the physical region to be formed. All arrays that contain values for each grid point in a block are therefore dimensioned from 0 to one greater than the maximum number of points allowed in the block. Thus the surrounding layer of points outside the block corresponds to $\xi^1 = 0$ on one side of the block and to $\xi^1 = \text{CMAX}(1, \text{block number})+1$ on the other:



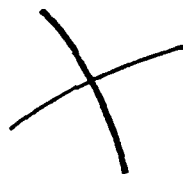
(Actually, provision is made for still another surrounding layer of points, corresponding to $\xi^1 = -1$ and $\xi^1 = CMAX+2$, in order to provide connections for use in partial differential equation codes using two-point one-sided differences. Since this second surrounding layer is not involved in the grid generation, no further account will be taken of its presence in the present discussion.)

2. Block Interfaces

The grid can be generated such that the grid lines cross the interface from one hexahedron to the next with complete continuity,



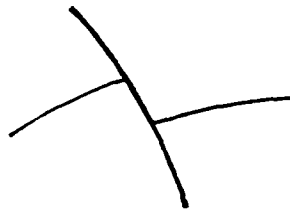
with slope continuity,



with only line continuity



or discontinuously.



With any degree of continuity, i.e., in all but the last case, adjacent blocks must, of course, have the same number of points on their common interface.

In the case of complete continuity, the interface is a branch cut, and the code establishes a correspondence across the interface using the surrounding layer of points outside the blocks. This allows points on the interface to be treated just as all other points, so that there is no loss of continuity. The physical location of the interface is thus totally unspecified in this case.

The case of slope continuity is accomplished simply by requiring the grid lines to intersect the interface orthogonally on both sides. This can be done either through Neumann boundary conditions, in which case locations of the points on the interface are determined by the code (with the shape of the interface specified by input), or by iterative adjustment of the control functions with the points on the interface specified by input.

Line continuity requires only that the same physical points be specified on the interface on each of the two blocks it joins, so that the points on the interface are completely specified by input. No continuity at the interface requires nothing at all, of course, and the adjacent blocks do not even have to have the same number of points on the interface in that case.

3. Fundamental Arrays

In the following discussion the field arrays (which contain values at each grid point in a block), such as R given below, include the block number as a subscript. The code actually operates with data from only one block at a time in these arrays and hence this subscript is always unity in the code. The present explanation of usage is greatly simplified by the inclusion of the block number as a subscript, however, and understanding of the actual operation is not necessary for usage. In any case the actual operation is discussed in Part II.

The three Cartesian coordinates x_i ($i = 1, 2, 3$) of the grid points in a block are in the real array `arrablock` are in the real array

$R(i, \text{block number}, \xi^1, \xi^2, \xi^3) \quad i = 1, 2, 3$

where (ξ^1, ξ^2, ξ^3) are the three curvilinear coordinates of the grid point in the computational block. The index i is usually referred to in the code as the integer CI when it identifies a curvilinear coordinate, e.g. ξ^i , and as the integer RI when a Cartesian component is identified, e.g. x_i . Similarly the integers $C1, C2, C3$ or $C(1), C(2), C(3)$ are used for the arguments ξ^1, ξ^2, ξ^3 . The block number is usually given as the integer B (or $NBLK$ in the subroutines).

Each grid point in a block is given a classification set in the integer array

$TYPE(\text{block number}, \xi^1, \xi^2, \xi^3)$

This array, which is set up by the code from the input, contains at each grid point one of the following alphanumeric values (the default is "FIELD" except on the surrounding layer where the default is "OUT")

TYPE = "FIX": indicates a point for which the Cartesian coordinates are not to be changed, e.g., a fixed point on a physical boundary.

TYPE = "FIELD": indicates a grid point for which the Cartesian coordinates are to be calculated by the grid generation system, e.g., a general field point.

TYPE = "IMAGE": indicates an image point, i.e., a point on a block boundary or surrounding layer of points, for which the Cartesian coordinates will be kept equal to those at another (object) point in the same or another block.

TYPE = "REFLECT": indicates a point on the surrounding layer which is the mirror-image reflection in a plane physical boundary of a grid point just inside the boundary.

TYPE = "AVERAGE": indicates a special grid point on a block boundary which is the average of all the adjacent grid points.

TYPE = "NEUMANN": indicates a grid point on a boundary at which the Neumann boundary conditions are to be applied. These boundary conditions may be either orthogonality or zero curvature. (Such a point moves along the boundary.)

TYPE = "ORTHOG": indicates a grid point on a boundary at which grid lines are to be made orthogonal to the boundary by iterative adjustment of the control functions. (This leaves the boundary point fixed.)

TYPE = "OUT": indicates a point completely out of the computation, e.g., inside a body in the interior of a block.

The correspondence across the interfaces between the hexahedrons in the physical region is established in the integer array

$$\text{IMAGE}(_, \text{block number}, \xi^1, \xi^2, \xi^3)$$

where the first subscript assumes the values 0,1,2,3 as explained below.

The Cartesian coordinates of points having TYPE = "IMAGE" (an image point) are kept equal to those of some other (object) point in the same or another block. The block number and curvilinear coordinates (ξ^i) of this object point are in the array IMAGE, where

$$\begin{array}{l} \text{image point} \\ \hline \text{object point} \left[\begin{array}{l} \text{block number} = \text{IMAGE}(0, \text{block number}, \xi^1, \xi^2, \xi^3) \\ \xi^1 = \text{IMAGE}(1, \text{block number}, \xi^1, \xi^2, \xi^3) \\ \xi^2 = \text{IMAGE}(2, \text{block number}, \xi^1, \xi^2, \xi^3) \\ \xi^3 = \text{IMAGE}(3, \text{block number}, \xi^1, \xi^2, \xi^3) \end{array} \right. \end{array}$$

Here the last four arguments of the array identify the image point, while the four values of the array identify the corresponding object point. This array is also set up by the code at input. As an example of the use of the IMAGE array, if TYPE(IB,IC1,IC2,IC3)= "IMAGE" then the

point with $\xi^1=IC1$, $\xi^2=IC2$, $\xi^3=IC3$ in block IB is an image point. The corresponding object point, say $\xi^1=C1$, $\xi^2=C2$, $\xi^3=C3$ in block B, is obtained from the IMAGE array as

```
B = IMAGE(0,IB,IC1,IC2,IC3)
C1 = IMAGE(1,IB,IC1,IC2,IC3)
C2 = IMAGE(2,IB,IC1,IC2,IC3)
C3 = IMAGE(3,IB,IC1,IC2,IC3)
```

Then the Cartesian coordinates at the image point are set equal to those at the object point by

```
R(1,IB,IC1,IC2,IC3) = R(1,B,C1,C2,C3)
R(2,IB,IC1,IC2,IC3) = R(2,B,C1,C2,C3)
R(3,IB,IC1,IC2,IC3) = R(3,B,C1,C2,C3)
```

(The notation IC1,IC2,IC3, or IC(1),IC(2),IC(3), for the curvilinear coordinates ξ^1, ξ^2, ξ^3 of an image point is common in the code, as is IB for the image for the block number.)

4. Scratch Files

The code uses files 7-10 as scratch files.

5. Adjustable Dimension Parameters

There are several dimension parameters that are set by identical PARAMETER statements in the main program and in the subroutines. These parameters can be changed by global edits.

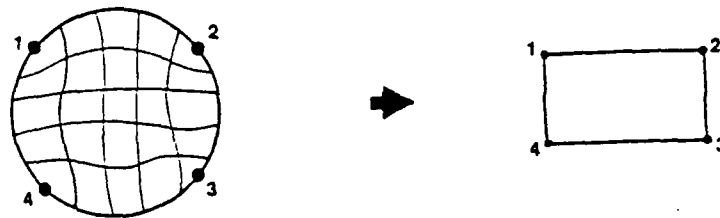
A resource summary is printed in order to allow the user to adjust the dimension parameters to reduce the storage required as much as possible. This summary shows the storage allotted and that actually used, as related to each of the adjustable parameters. Some of the resources given are affected by more than one of the parameters. A complete list of these adjustable parameters is given in Section II-A2.

B. SET-UP PROCEDURE

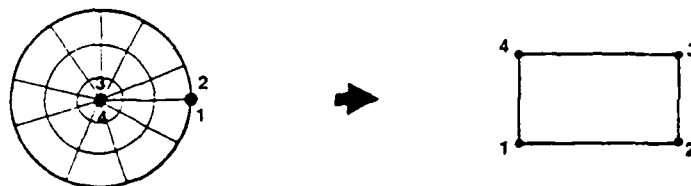
1. Block Structure

Setting up the code to generate a grid consists of first determining the basic structure for the hexahedrons(blocks) that are to fill the physical region. This amounts basically to deciding where the eight corners (four in 2D) of each hexahedron will be located in the physical region.

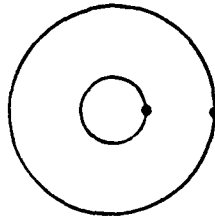
Thus a circular region in 2D could be filled with one block having all four corners on the circle:



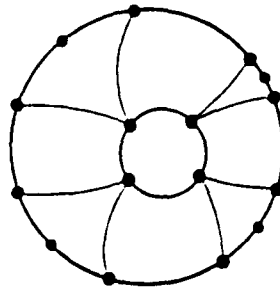
or with one having two corners coincident on the circle, and the other two coincident on an interior point:



Similarly, a 2D annular region could be filled with a single block:



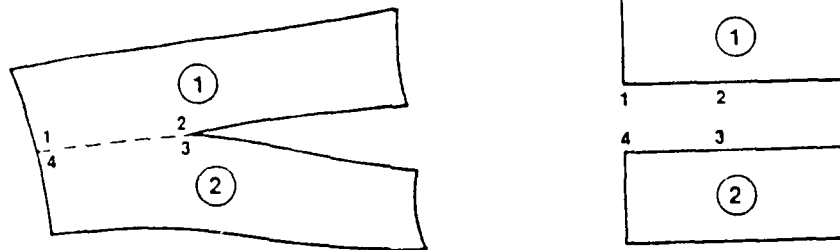
or perhaps with eight blocks:



Other possibilities also exist, of course. (The above figure could be a nine-block filling of the empty circular region mentioned first.) More complicated and three-dimensional cases are illustrated in the examples given in Vol. I.

2. Boundary Points

After block corners have been located, the next step is to decide on the number of grid points that are to occur in each direction on each side of the block. This information on the size of each block is input to the code as described in Sect. I-C2. Here certain considerations must be met regarding adjacent blocks if the grid lines are to be continuous across the interfaces between blocks. Thus with a configuration of the form



Block 2 must have the same number of points from 4 to 3 as Block 1 has from 1 to 2 if the grid lines are to be continuous across the interface. (If line continuity is not desired, no such restrictions apply.)

The next step is to place the desired number of points on each side of the block that corresponds to a segment of a physical boundary. This involves specifying the Cartesian coordinates of each such point. These points can be read into the code either from a file or directly from the

input (Sect. I-C3). The placement of these boundary points can be done using the front-end code (Vol. II), which generates boundary surfaces and prepares files in the proper format for input to the grid code. In any case the required file format is discussed in Sect. I-C3. (The grid code can be run in the 2D surface mode to generate points on a generally curved boundary surface for later input for a 3D region. Such 2D results can also be passed back through the front-end code for transformation, e.g. translation, rotation, and scaling, before being input as a boundary segment to the 3D code.)

3. Interfaces

The grid lines across the interfaces between blocks can be completely continuous, can have continuous slope, can be simply continuous but with a break in slope, or can be discontinuous (Sect. I-A2). This provision is made on the input as described in Sect. I-C7 for each interface. Although these interfaces are not physical boundaries, and in fact may be completely determined by the code, it is necessary to establish an initial point distribution on all six sides of each block (four in 2D) when elliptic generation is used in order for the code to determine the control functions. This initial point distribution on the interfaces can be read in as are points on physical boundary segments, (Sect. I-C3) or can be interpolated (Sect. I-C6) by the code from values read in on the edges of the interfaces. Even the values on the edges can be interpolated if desired, leaving only the corner values to be read in. In most cases, however, it will be preferable, in the interest of good control of the grid, not to rely on interpolation for edge values.

4. Grid

The code can function as an algebraic generation system based on various forms of transfinite interpolation using either Lagrange or Hermite interpolation, the latter providing boundary orthogonality and specified off-boundary spacing, or can be run as an elliptic generation system with control functions determined automatically from the boundary point distribution or iteratively to establish boundary orthogonality with specified spacing off the boundary. Neumann or reflective boundary conditions can also be called for. Different boundary treatments can be used on different segments of the boundary.

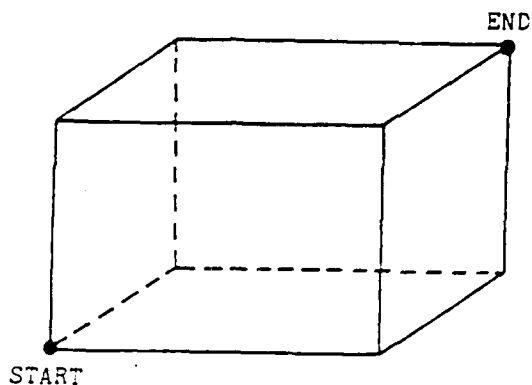
Finally, various forms of output can be called for, e.g. print out and/or storage for plotting all or specified sections of the grid, or file storage of the entire grid in various formats for input to a PDE code.

5. Input

The input to the code is accomplished by successive reads of a NAMELIST called INPUT, followed by successive reads of another NAMELIST called OUTPUT. The former initializes the code for the grid generation, and the latter specifies the type of output to be provided. All features of the input are discussed in detail in Sections I-C and I-D.

In the input it is convenient to identify rectangular sections in a block for the purpose of reading in Cartesian coordinate values of boundary points, for classifying points as discussed above, and for establishing correspondences across interfaces. This is accomplished

through the integer arrays `START(3)` and `END(3)`, which contain the three curvilinear coordinates of two diagonally-opposite corners defining the rectangular section of the block:



Thus we have $\xi^i = \text{START}(i)$, $i=1,2,3$ as the coordinates of one corner of the section, and $\xi^i = \text{END}(i)$, $i=1,2,3$ for the other. (Surface sections will have $\text{START}(i) = \text{END}(i)$ for one value of i , while line sections will have this equality for two values of i .) Any entry in `START` may exceed the corresponding entry in `END`, the progression being from `START` to `END` in any case.

With the numbered-points mode (Section I-C23), single entries for `START` and `END`, and also for `ISTART` and `IEND`, will be taken as point numbers on all input statements involving these quantities. Negative values for entries of `START`, `END`, `ISTART`, or `IEND` indicate a point number (Section I-C23) if one with that number has been set, or may indicate a stored value (Section I-C1).

C. NAMELIST/INPUT/

The grid generation is set up by successive reads of a NAMELIST called INPUT. Each of these reads is done by an input statement of the form

\$INPUT ITEM = "__", --- \$

with the quantity ITEM identifying what type of input data is given on each particular read of INPUT, as discussed below. The relevant input quantities also appear on this statement, separated by commas or blanks, in the form

quantity = value

Arrays appear as

quantity = value, value,--

and repeated values can be indicated as

quantity = N*value

where the number, N, preceding the asterisk indicates the number of repeats. Single array entries can appear simply as quantity = value, if the first entry is intended, or as quantity(i) = value if the i-entry is intended. The NAMELIST variables are defaulted again after each input statement. In all cases only relevant variables to be changed from default values need appear on the input statement. The code checks for missing or unreasonable values of required quantities. Also when only the first two entries in a 3D array are relevant (e.g., for 2D) only two values need be given. This setup portion of the input is terminated by an input statement with ITEM = "END".

In the following discussion of input, upper-case letters are to appear on the input statements exactly as given here, on either side of the equal signs and in or out of quotes. Lower-case letters, or underlined blanks, indicate values to be given for quantities, and are to be replaced by integers or real numbers as appropriate. The exponential notation can be used for real numbers when desired. Real numbers that happen to be integral can be given as integers if desired.

1. Calculation and Storage of Input Parameters for Later Use

Values of SIZE, START, END, ISTART and IEND can be calculated from sums or products of values and stored for use on later input statements. This is done by an input statement with ITEM = "SETVAL", with the type of calculation specified by MATH:

$$\text{"SUM": } \sum_1 \text{ TERMS}(i)$$

$$\text{"SUM-1": } \text{TERMS}(1) + \sum_1 [\text{TERMS}(i) - 1]$$

$$\text{"DIF": } \text{TERMS}(1) - \text{TERMS}(2)$$

$$\text{"DIF+1": } \text{TERMS}(1) - \sum_1 [\text{TERMS}(i) + 1]$$

$$\text{"PRODUCT": } \prod \text{ TERMS}(i)$$

The terms in the calculation are input in the integer array TERMS, and the storage location as the integer VALOUT.

These stored values are invoked on any later input statements by giving a negative value for the intended quantity, the magnitude being the storage location. For example, the following input statement stores the sum of 3, 4, and 8 in storage location 3 for later use:

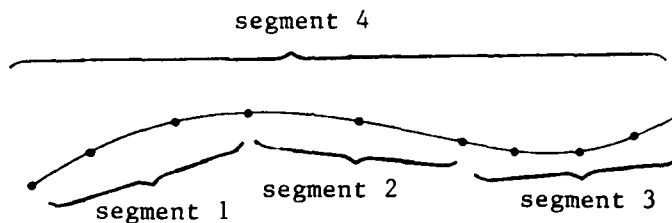
\$INPUT ITEM = "SETVAL", TERMS = 3,4,8, MATH="SUM", VALOUT = 3 \$

Then on a subsequent input statement, the usage

SIZE = 10, -3, 8

would assign the values 10, 15, 8 to SIZE.

The SUM-1 and DIF+1 options are particularly useful in setting the number of points on a segment from the number on component segments. Thus, in the example below



the number of points (10) on segment #4 can be calculated from the number of points on segments #1, #2 and #3 with TERMS=4,3,5 and MATH="SUM-1". Similarly, the number of points (3) on segment #2 can be calculated from the numbers of points on segments #1, #3 and #4 with TERMS=10,4,5 and MATH="DIF+1".

Entries in TERMS may now themselves be stored values, indicated by a negative entry. Note that this means that it is not possible to explicitly use negative values in the calculation, rather "DIF" must be used to perform subtraction.

The default for MATH is "SUM-1", except when only a single entry of TERMS is given in which case the default is simply to store the value.

Thus in the above example, the number of points on each segment could first be stored against the segment number by

\$INPUT ITEM = "SETVAL", TERMS = 4, VALOUT = 1 \$

\$INPUT ITEM = "SETVAL", TERMS = 3, VALOUT = 2 \$

\$INPUT ITEM = "SETVAL", TERMS = 5, VALOUT = 3 \$

Then the number on segment #4 can be set from these other segments by

\$INPUT ITEM = "SETVAL", TERMS = -1,-2,-3, VALOUT = 4 \$

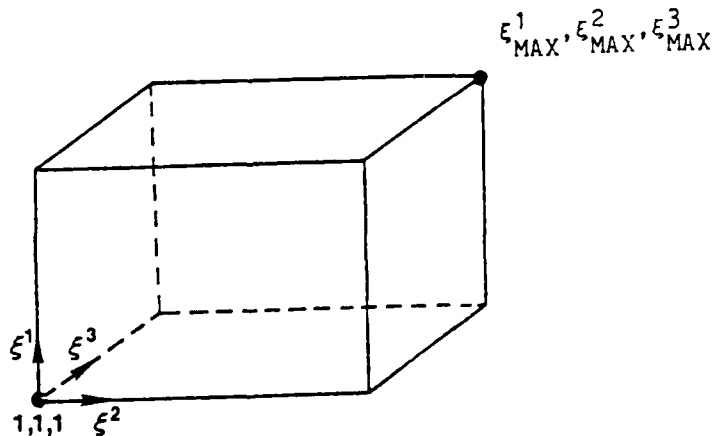
Alternatively, with the number of points set on segments #4, #1 and #3, the number on #2 can be set with TERMS=-4,-1,-3. Positive (actual numbers) and negative (storage locations) entries in TERMS can be mixed.

Negative values for entries of START,END,ISTART, or IEND now indicate a point number (Section I-C23) if one with that number has been set. The feature described in Section I-C1 thus must not use a point number as a storage location (VALOUT).

2. Block Specifications

The size of each block must be specified by an input statement with ITEM ="BLOCK", giving the block size as three entries in the integer array SIZE(3), e.g.

\$INPUT ITEM = "BLOCK", SIZE= $\xi_{MAX}^1, \xi_{MAX}^2, \xi_{MAX}^3$ \$



before any other reference to that block is made. The blocks are numbered successively by the code as each input statement of this type is encountered. The three entries in SIZE will be placed in the array CMAX for the current block. One such input statement must be included for each block. The 2D mode is activated by the omission of the third entry in SIZE, in which case the grid will be in the x-y plane. Block numbers are remembered from one input statement to the next and hence can be omitted on subsequent statements as long as the same block is intended.

When the new numbered-points feature (Section I-C23) is used, the block size can be set from a numbered point by the input statement

$$\text{\$INPUT ITEM = "SIZE", SIZE = } \xi_{\text{max}}^1, \xi_{\text{max}}^2, \xi_{\text{max}}^3 \text{\$}$$

Here three entries (the actual block dimensions) can be given for SIZE, or a single number (taken as the point number) can be given. In the latter case, this point would be that at the block corner opposite the point at (1,1,1).

In the numbered-point mode, each block is introduced by an ITEM="BLOCK" without SIZE. The block size is set later, following the statements that set the numbered points and place the boundary segments (ITEM="POINT" and ITEM="SEGMENT", Section I-C23), by an ITEM="SIZE" statement.

All the boundary segments associated with the block can be read at once by including FILE, giving the file number, on the ITEM="BLOCK" statement. In this case the reading is done as described in the revision of Section I-C3, but for only a single block (cf. also Section

I-C23). The only types of input statements that are allowed between the ITEM="BLOCK" and ITEM="SIZE" statements are those that store values for later use (ITEM="SETVAL", Section I-C1), those that set point locations (ITEM="POINT", Section I-C23), and those that read boundary segments from a previously read entire file (ITEM="SEGMENT", Section I-C23).

3. Reading of Boundary Points

After the block size has been specified, the Cartesian coordinates of grid points on boundaries or interfaces on that block can then be read either from a file or can be included directly on the input statement.

Reading from a file is done by an input statement with ITEM = "FILE". Here the file number is specified by the integer FILE, the block is given by the integer BLOCK, and the section of points to be read is specified by the integer arrays, START(3) and END(3), as described in Section I-B5. (Note that "FILE" and FILE have different meanings, the former being an alphanumeric value given to the variable ITEM, while the latter is an integer variable, the value of which is the file number.) The file will not be rewound before being read unless REWIND="YES" appears on the input statement. It is, of course, not necessary to include REWIND on the initial reading of a file.

The progression of the reading of points in the section is specified by the integer array ORDER(3), the three entries of which are some permutation (not necessarily cyclic) of 1,2,3. The points are read with the curvilinear coordinate ξ^i , with i given by the first

entry in ORDER, running fastest, etc., on the section. (ORDER does not affect the order in START and END, i.e., the first entries therein are the section limits on the ξ^1 coordinate, etc.) The default for ORDER is 1,2,3.

In general, the file must contain a triad of real Cartesian coordinate values for each point in succession. The file must be unformatted and written as a sequence of triads of single real values, one point per line, i.e.,

```

      x y z
    - - - -
      x y z
    - - - -
      .
      .
      .

```

unless TRIAD = "NO" is included, in which case each value is on a separate line. If only two Cartesian coordinates are present on the file for each point, the file must have one coordinate to a line, and TRIAD="NO" and RORDER=1,2 must be included. (The 2D mode assumes that all three coordinates are present on the file unless TRIAD and RORDER are included in this manner.) The format is controlled by FORM to be unformatted, E20.8, or list-directed as FORM is equal to "UNFORM", "E", or "LIST". The default is unformatted.

The integer array RORDER(3), the three entries of which are some permutation (not necessarily cyclic) of 1,2,3, specifies the order in which the Cartesian coordinates of the point occur on the file, i.e.,

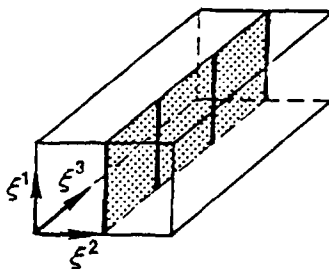
the coordinate x_i , with i equal to the first entry in RORDER, is given first, etc. The default for RORDER is 1,2,3. Zero entries in RORDER cause the Cartesian components that are thereby omitted not to be read. In this case the file must contain only the components that are to be read, i.e., less than three values for each point and the values must be given one to a line. (In 2D, the arrays START, END, ORDER, and RORDER all require only two entries, and only two Cartesian coordinate values are required on the file for each point.)

Since the entries of ORDER set the order of a nest of loops that read the points from the file (with the first entry indicating the innermost loop), it is only necessary to include ORDER when the two directions on the segment of the block are not 1,2, or 2,3, or 1,3, or if the faster direction is higher than the slower direction, e.g. 2,1. In any case, giving the faster running direction as a single entry for ORDER will suffice.

As an example of this reading of points from file, the input statement

```
$INPUT ITEM = "FILE", FILE = 12, BLOCK = 1, START = 1,1,1,
      END = 2,3,4, ORDER = 1,3,2, RORDER = 3,2,1 $
```

reads 24 points for block 1 from file 12. From START and END, the ξ^1 coordinate runs from 1 to 2, the ξ^2 coordinate runs from 1 to 3, and the ξ^3 coordinate runs from 1 to 4. From ORDER, the points are read on surfaces of constant ξ^2 , along lines on which ξ^3 is constant:



These points thus are read in the following order:

(1,1,1)
(2,1,1)

(1,1,2)
(2,1,2)

(1,1,3)
(2,1,3)

(1,1,4)
(2,1,4)

(1,2,1)
(2,2,1)

(1,2,2)
(2,2,2)

(1,2,3)
(2,2,3)

(1,2,4)
(2,2,4)

(1,3,1)
(2,3,1)

(1,3,2)
(2,3,2)

(1,3,3)
(2,3,3)

(1,3,4)
(2,3,4)

In this example the Cartesian coordinates for each point are read in the order $\underline{x_3}$, $\underline{x_2}$, $\underline{x_1}$, because of RORDER, and must appear on file 12 in that order, of course.

Similar provision is made for including the grid points directly on the input statement, this being done by an input statement with ITEM="LIST". (The arrays BLOCK, START, END, ORDER, and ORDER have the same functions as for reading from a file.) In this case the Cartesian coordinates are input as real values in the array VALUES on the input statement, e.g.

```
$INPUT ITEM = "LIST", BLOCK = 1, START = 1,1,1, END = 3,3,3,
```

```
VALUES =  __, __, __,
          __, __, __,
          __, __, __,
          __, __, __,
          __, __, __,
          __, __, __,
          __, __, __,
          __, __, __,
          __, __, __,
          __, __, __ $
```

Here the first three entries in VALUES are the Cartesian coordinates of the first point, the second triad gives the coordinates of the second, etc. As with a read from a file, zero entries in RORDER mean that fewer than three values appear in VALUES for each point. (In 2D, with only two entries in RORDER, there need be only two entries per point in VALUES, i.e., the third column in the above example need not appear.)

In the numbered point-numbered segment mode (Section I-C23) an entire file of boundary segments is read in at once by including ALL="YES" on an ITEM="FILE" statement instead of BLOCK,START,END,ORDER,

and RORDER. These segments may be associated with several blocks. In this mode the three Cartesian coordinates must be placed as a triad on a line of the file, and all three must be present, even in the 2D mode.

The individual segments are then put into place by input statements with ITEM="SEGMENT", including BLOCK, START, and END, and perhaps ORDER, RORDER, CLASS, and IPRINT. In addition, the segment number must be given on this statement as SEGMENT, corresponding to the COREOUT number associated with the segment in the input of the front-end code (Vol. II, Section I-E21). The order in which the segments are put into place here does not have to conform to that on the file, and segments can be placed in more than one position on the block (e.g. cuts) if desired.

It is also possible, when reading points either from a file or directly on the input statement, to specify the classification (Section I-A3) of the points on the section being read by including

CLASS = "___"

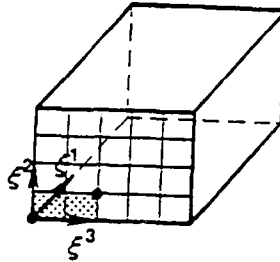
where one of the values of TYPE given in Section I-A3 appears in the quotes. Finally the points will be printed if IPRINT = "YES" is included.

4. Fixed Points

Since all points in a block are defaulted to "FIELD" points, it is necessary to explicitly designate fixed boundary points as "FIX". This can be done as the points are read by including CLASS="FIX" on the input statement. Alternatively, this classification can be done for any section of points through a separate input statement with ITEM =

"FIX" and including BLOCK, START, and END to identify the section. For example, the following input statement classifies the 30 points on the $\xi^1=1$ boundary of block 1 as fixed:

```
$INPUT ITEM = "FIX", BLOCK = 1, START = 1,1,1, END = 1,5,6$
```



Once a point has been designated a fixed point, the classification cannot be changed, except by the "UNFIX" operation or in case of special points (Section I-C12). It should be noted that the default classification is "FIELD", and hence fixed foundry points must be classified as "FIX" either explicitly or as read or interpolated.

Points can be "unfixed" by an input statement with ITEM="UNFIX". The classification is returned to the default, "FIELD". The usage is as with ITEM="FIX". The "UNFIX" statement is the only statement that affects "FIX" points.

This feature can be used to classify all the sides of a block as "FIX" by first using ITEM="FIX" for the entire block and then using ITEM="UNFIX" for the interior.

5. Out Points

Similarly, points on a section not already designated "FIX" can be designated as "OUT", i.e., to be ignored completely, by an input statement with ITEM="OUT", and including BLOCK, START, and END to identify the section. (Points on the surrounding layer outside a block are defaulted to "OUT".)

6. Section Interpolation

Values of the Cartesian coordinates for grid points on any section of a block can be interpolated from already-specified values on the section boundary by transfinite interpolation through an input statement with ITEM="INTERP" and including FUN="POINTS". The interpolation will not alter the values at points in the section that have been already classified "FIX". Classification can be called for on the interpolation input statement by including CLASS="FIX", etc., but is done after the interpolation.

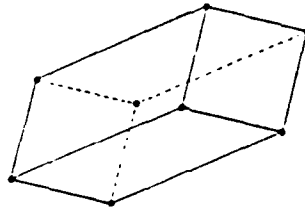
If insufficient boundary values are given for section interpolation, sub-blocks or for the interpolation for the algebraic grid, the code will attempt to supply the missing values by transfinite interpolation of lesser dimensionality, using "ARC" blending functions for missing face values and "LINEAR" blending functions for missing edge values.

Also, section interpolation will be automatically delayed until the input is completed if values have not been set on the entire section boundary at the time when the interpolation is invoked.

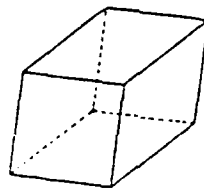
This interpolation can be used to set points on boundaries for which the actual shape is not important, e.g. remote boundaries at 'infinity' or initial values on interfaces between the hexahedrons in the physical region. (It is necessary to set points on the interfaces even when these points will be changed in the course of the grid generation since the code requires initial points on all six sides of a block to determine the control functions, Section I-C15.) Although this interpolation can also be used to evaluate control functions on the section, this use is not standard and was included only for completeness to provide for unusual cases. (FUN="CONTROL" interpolates control functions. In that case, read 'control functions' for 'Cartesian coordinates' in the present section.)

The section on which the interpolation is to be done is identified by BLOCK, START and END, and the form of the interpolation is set by PROTYP="___", where the entry is "FACES", "EDGES", or "CORNERS", corresponding to the portion of the section boundary to be matched by the transfinite interpolation. Cartesian coordinate values for all points on the section boundaries that are to be matched must have been set by previous input statements. The code checks for unspecified required boundary values.

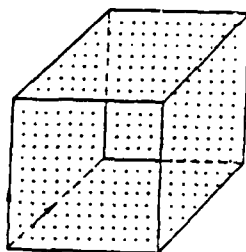
With PROTYP="CORNERS", the interpolation is from the eight corners only (four corners in 2D):



In this case Cartesian coordinate values at only the eight corner points need be specified before the interpolation. Values of the Cartesian coordinate values at all other points in the section, including its boundary, that have not been classified "FIX" will be set by the interpolation. With "EDGES", the interpolation is from the twelve edges only (same as "CORNERS" in 2D):

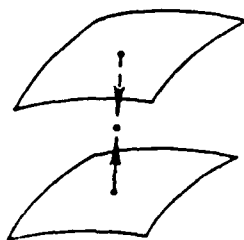


Here values for all points on the twelve edges must be set prior to the interpolation input statement. Again Cartesian coordinate values will be set by the interpolation at boundary points, as well as at interior points, of the section, except on the twelve edges. Complete transfinite interpolation from the entire boundary occurs with "FACES" (the default):

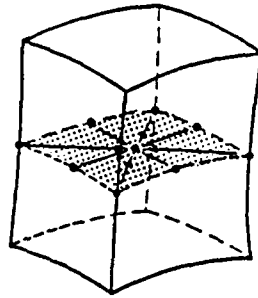


for which values for all points on the entire section boundary must be set before the interpolation. Here the interpolation sets only values in the interior of the section. The omission of PROYP produces this latter form of interpolation by default.

It is also possible to restrict the interpolation to less than the full dimensionality of the section by including `PRODIR = __, __` where the integer entries specify the curvilinear coordinate directions in which the interpolation is to be performed. (The omission of PRODIR produces interpolation to the full dimensionality possible on the section.) With only one entry in PRODIR, the interpolation is simply along a straight line between two boundary points:

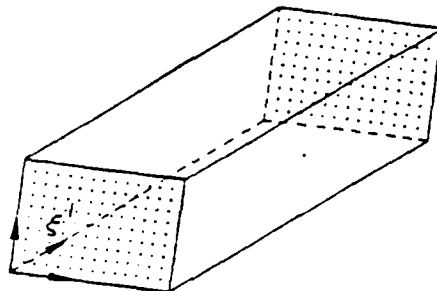


Two entries in PRODIR gives 2D interpolation even in a 3D section, and PROTOP functions on surfaces on which the two entries in PRODIR vary in the manner described above for 2D.

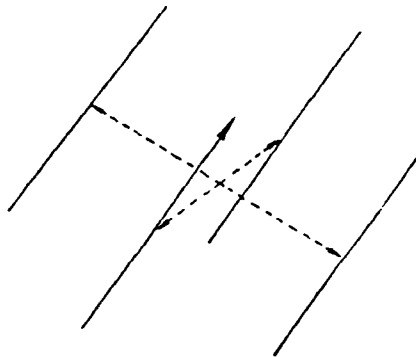


If the section is two-dimensional, it is not meaningful to include PRODIR with more than one entry.

This restriction of the dimensionality of the interpolation reduces the number of points that are matched on the section boundaries, of course. For example, in 3D with PROTOP="FACES" and PRODIR=i, the interpolation is from the two faces on which ξ^i is constant, instead of from all six faces:



Therefore values must be set on only these two faces before the interpolation. The interpolation will set values on the remaining four faces, as well as inside the section, in this case (The effect is similar in 2D). With `PROTYP="EDGES"` and `PRODIR=1`, the interpolation is from the edges on which ξ^1 varies (This has no meaning in 2D):



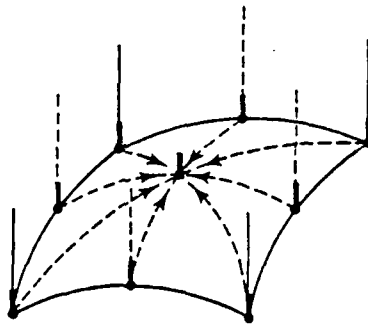
The use of `PRODIR` with `PROTYP="CORNERS"` is not meaningful. (`PRODIR` is defaulted to 1,2,3, i.e., to interpolation in all directions. This automatically reduces to 1,2 in 2D.)

It must be noted that the entries in `PROFOR` and `BLEND`, discussed below, are with reference to the three curvilinear directions regardless of the entries of `PRODIR`. Thus if `PRODIR = 3`, indicating interpolation in only the ξ^3 direction, a value for `PROFOR` must appear as `PROFOR(3)=value`.

The interpolation may be either Lagrange or Hermite, individually in each direction, as specified by the values given by `PROFOR="___",__,_"`. (In 2D, only two values need be given.) The choices are as follows:

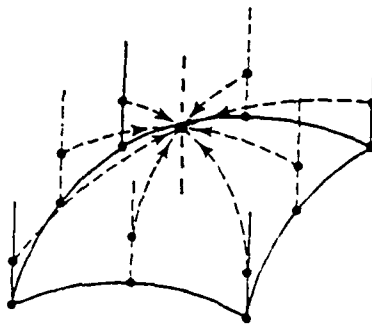
- "LAGRANGE" - Lagrange (values specified on each end). The default.
- "HERMITE" - Hermite (values and spacing specified on each end).
- "HERMITE1" - Incomplete Hermite (values specified at each end, spacing specified at first end only).
- "HERMITE2" - Incomplete Hermite (values specified at each end, spacing specified at second end only).

For the Hermite cases, the slope is made orthogonal to the boundary with a spacing determined either through specification by inclusion of SPAVAL = spacing, or through Lagrange transfinite interpolation from the section sides if SPAVAL is omitted:

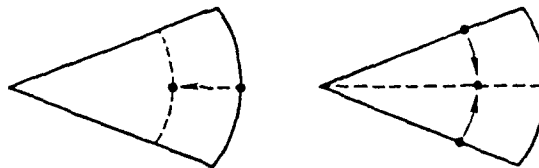


The omission of PROFOR produces Lagrange interpolation by default.

Finally, the blending functions for the interpolation will be linear if BLEND="LINEAR", or will be based on an interpolated arc length distribution constructed from the point distribution on the section boundary if BLEND="ARC" (the default). The arc length distribution used in the latter case is determined by transfinite interpolation from the four faces (in 3D) on which the curvilinear coordinate in the blending function varies:



(This interpolation for arc length is automatic in the code and is not dependent on input.) For sections having a degenerate boundary, i.e., one on which the Cartesian coordinates for all points in one or more directions are the same (as for a polar axis, etc.), the degenerate distribution will be ignored in the interpolation for the blending functions based on arc length:



If degeneracy occurs on all the boundaries needed for the arc length interpolation, the blending functions will default to linear.

It should be noted that the boundaries of a section on which interpolation is used do not have to be actual physical boundaries or even interfaces between blocks. For example, with the configuration



and points inside the notch designated "OUT", 1D interpolation could be used first to set values on the two dotted lines, and then 2D interpolation could be used to set values in the three sections, assuming that boundary values have been read on the solid lines. Alternatively, values could be read on the dotted lines, without designating the points thereon as "FIX", followed by only the 2D interpolation in the three sections.

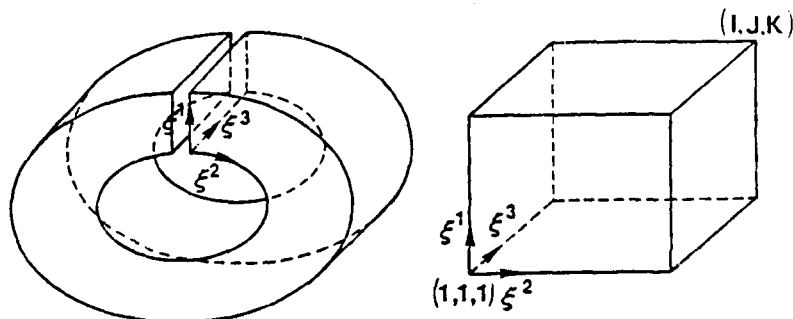
7. Interface Linkage

If complete continuity of grid lines across the interfaces between blocks is desired, the linkage among the various blocks must be specified by input statements with ITEM = "CUT", including BLOCK, START and END to identify an object section of grid points which is to be linked to an image section identified by IBLOCK, ISTART and IEND. Values in START and ISTART may exceed those in END and IEND, and must be set so that the progression from ISTART and IEND on the image section corresponds to that from START to END on the object section. Also included is ORDER to specify the order in which the points on the object section are to run, the order on the image section being fixed as 1,2,3. (cf.

Section I-C3 for an explanation of ORDER, which is defaulted to 1,2,3). Since the classification of "FIX" points will not be changed, the cut section may include "FIX" points. Points on the object face that correspond to points classified as FIX, ORTHOG, or NEUMANN on the image face will be made image points. It is thus not necessary to classify and read data for such corresponding points separately.

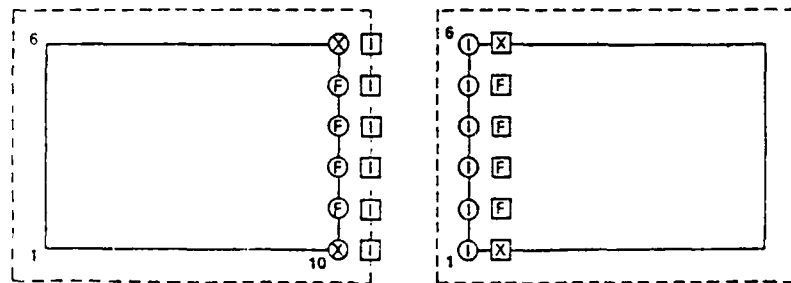
One such input statement must appear for each pair of interface sections to be linked. The points on the object section that have not previously been designated "FIX" will be classified automatically by the code as "FIELD", and the elliptic grid generation system will operate at those points exactly as in the rest of the field. The points on the image section will be classified automatically as "IMAGE", so that the coordinate values at those points will be kept equal to the values at the corresponding object points. Also, a layer of points outside the object section will be classified automatically as "IMAGE", and these points will be coupled by the code with corresponding object points just inside the image section, with values on the former being kept equal to values on the latter. This image layer extends one point beyond each edge of the cut section. A similar layer will be set up outside the image section.

For example, consider the 0-type configuration shown below:



```
$INPUT ITEM = "CUT", START = 1,1,1, END = I,1,K,  
ISTART = 1,J,1, IEND = I,J,K $
```

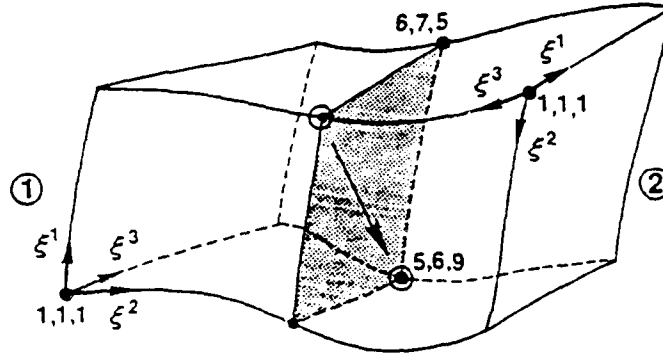
Two blocks can be linked as follows:



Here the circled points are coupled, and the squared points are coupled, with F, X, and I indicating "FIELD", "FIX", and "IMAGE" points, respectively, on the diagram. The input statement for this example is as follows:

```
$INPUT ITEM = "CUT", BLOCK = 1, START = 10,1,
      END = 10,6, IBLOCK = 2, ISTART = 1,1, IEND = 1,6 $
```

It is not necessary that a cut section cover an entire side of a block; just that the size of the object and image sections be the same. It is also not necessary that the same curvilinear coordinates vary over the object and image sections, or that the directions of variation be the same, since the correspondence between coordinate species on the two actions can be established via ORDER. For example, the correspondence of interfaces for the blocks shown below



is established by the input statement

```
$INPUT ITEM ="CUT", BLOCK = 1, START = 6,7,1 END = 1,7,5
      IBLOCK = 2, ISTART = 1,1,9 IEND = 5,6,9, ORDER = 3,1,2 $
```

Since cuts in multiple-block configurations may include the edges of a block side, it is not necessary to give a separate input statement for each edge. Statements for edges in 3D, or for points in 2D, cause the image-object correspondence to be set up only between the two indicated edges or points, without any surrounding layers.

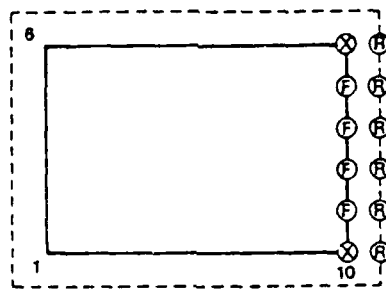
Although values of the Cartesian coordinates at points on a cut will be determined by the grid generation system, it is necessary to have initial values on the cut in order to evaluate the control functions and to start the iteration. Therefore initial values on the cut must either be read or set by interpolation. The reading of values on a cut is done as on an actual boundary (Section I-C3), except that the points must not be designated "FIX". Interpolation is as described in Section I-C6.

The "CUT" statements should be grouped at the last of the input. (Cuts within a block should be grouped at the end of the input statements for that block.) This is so that all of the point classifications will have been made before the cuts are set up.

8. Reflective Boundary

A plane reflective boundary can be established by an input statement with ITEM="REFLECT", using BLOCK, START, and END to identify the section. (This feature is only applicable to plane boundaries.) This causes the points on the section not previously designated "FIX" to be designated by the code as "FIELD" points, and the adjacent outside points on the surrounding layer to be designated as "REFLECT". As with a cut, this adjacent layer is made to extend one point beyond the edges of the reflective section on all sides. The reflective section, however, must not include the edges of a block unless the edge is part of a cut section on another side since the code must calculate a normal vector at all points on the section. It is permissible to include "FIX" points in the interior of the section, and the classification of such points will not be changed.

An example of a reflective section follows:



The input statement for this illustration is

\$INPUT ITEM = "REFLECT", BLOCK = 1, START = 10,2, END = 10,5 \$

Note that the designation of the reflective section is on a block side on the input. The code designates the appropriate points on the surrounding layer as "REFLECT". A point designated "REFLECT", being outside a boundary, has its Cartesian coordinate values kept equal to a mirror-image reflection of those at the corresponding grid point across the boundary just inside the block.

Although the Cartesian coordinate values for points on a reflective boundary will be determined by the elliptic grid generation system, initial values must be set either by reading values (Section I-C3) or by interpolation (Section I-C6).

9. Boundary Orthogonality through Neumann Boundary Conditions

Orthogonality on a boundary section can be called for by an input statement with ITEM="NEUMANN", with BLOCK, START, and END to identify the section. Here all points on the section not already designated as fixed points will be designated as points at which orthogonality will be enforced through Neumann boundary conditions. These points thus will move along the boundary. Again initial values must be set by reading (Section I-C3) or by interpolation (Section I-C6). Boundary orthogonality is achieved by the code splining a section of the surface and locating the point on the spline from which a normal extends to the first point off the surface.

The off-boundary spacing can be set on the entire section by including SPAVAL, or on portions by the operation ITEM="SPACE", as in Section I-C9. The default is also as in that section.

The section to be splined can be made to extend beyond that of the "NEUMANN" points by including ISTART and IEND to define the splined section. If more than one "NEUMANN" section appears on a single splined section, NEW="NO" must appear on all ITEM="NEUMANN" statements after the first that refer to the same splined section. If ISTART and IEND are omitted, the section to be splined defaults to that specified by START and END.

In the following illustration:

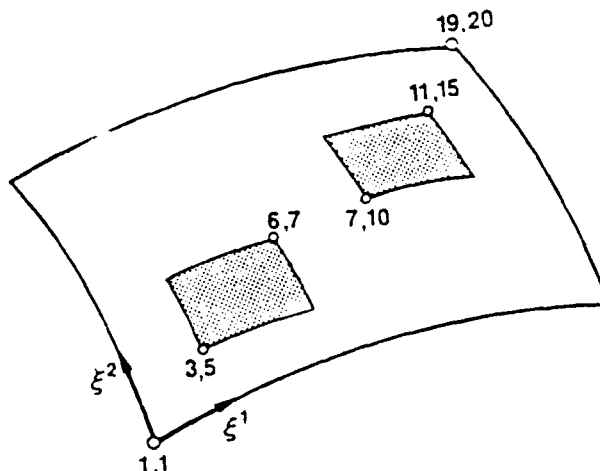
```
$INPUT ITEM = "NEUMANN", BLOCK = 1, START = 3,5, END = 6,7,
```

```
ISTART = 1,1, IEND = 19,20 $
```

```
$INPUT ITEM = "NEUMANN", BLOCK = 1, START = 7,10, END = 11,15,
```

```
NEW = "NO" $
```

two sections are specified for boundary orthogonality, and a larger section that includes both is splined:



The Neumann boundary points can be restricted to move along only one family of curvilinear coordinate lines on all, or a portion of, a Neumann section by the use of ITEM = "NEUMAN1", "NEUMAN2", or "NEUMAN3" after the Neumann section has been set up by the ITEM = "NEUMANN" statement. These three input statements serve to restrict the movement to along the curvilinear coordinate direction indicated by the name. The boundary portion is indicated by BLOCK, START, and END as usual. The Neumann section on which this portion lies must have been set up before this restriction of direction is made.

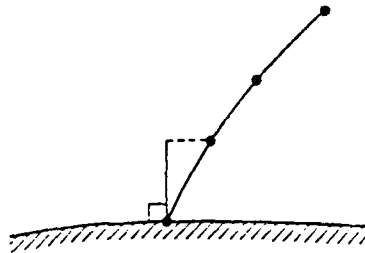
Neumann boundary conditions can also be used to impose zero-curvature extrapolation as described in Section I-C26.

10. Boundary Orthogonality through Control Functions

Orthogonality on a boundary section can also be called for by an input statement with ITEM="ORTHOG", with BLOCK, START, and END to identify the section. Here all points on the section not already designated as fixed points will be designated as points at which orthogonality will be enforced through iterative adjustment of the control functions. The section defined for this purpose must not include block edges unless the edge is part of a cut section on another side. The section may include "FIX" points in the interior, and the classification of such points will not be changed.

In contrast to the use of Neumann boundary conditions, the points in this case do not move on the boundary. The off-boundary spacing on the entire section can be set by including SPAVAL. (If this spacing is to be different on different parts of the section, ITEM="SPACE" can be

used to set the spacing on each part, Section I-C11). If SPAVAL is omitted, the spacing defaults to that from the algebraic grid, but projected normal to the boundary:



The extent of the orthogonality into the field is controlled by the values given for CONFAC (defaulted to 1.0). A value less than 1.0 increases the extent. The interval at which the control functions will be updated is set by including the integer CONUPI (the default is 1, i.e., update at each iteration). This updating is not started until 10 iterations have been completed. In general, optimum acceleration parameters (the default) should be used when the control functions are iteratively adjusted. Again initial values must be set on the section by reading (Section I-C3) or by interpolation (Section I-C6).

The section does not have to be on a block side. If the section is not on a block side, the side of the section on which the orthogonality is to be applied will be the upper side unless DIRECT="LOWER" is included. Thus if orthogonality is intended on both sides of a section in the interior of a block, it is necessary to use two input statements of this type, one for each side.

If INTORT="YES" is included, such internal "ORTHOG" points will be reclassified as "FIELD" after the grid has been generated and before the output file is written.

11. Off-Boundary Spacing

The off-boundary spacing for boundary orthogonality or Hermite interpolation can be set by an input statement with ITEM = "SPACE", section defined by BLOCK, START, and END and the spacing given by SPAVAL.

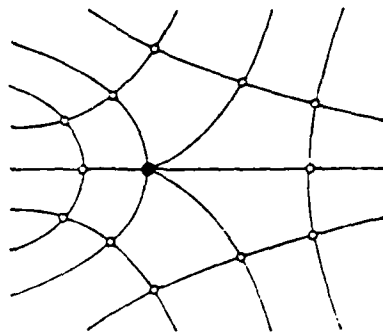
12. Special Points

Special points in the field can be made averages of all of the neighboring field points by an input statement with ITEM = "AVERAGE", with BLOCK and the array POINT(3) identifying the point. Here POINT contains three integer entries giving the curvilinear coordinates of the special point. (This designation will override a previous "FIX" designation.) If WEIGHT="YES" is included on the input statement, then the average will be weighted with the cell volume, i.e.,

$$\underline{r} = \frac{\sum_{\ell} V_{\ell} \underline{r}_{\ell}}{\sum_{\ell} V_{\ell}}$$

where V_{ℓ} and \underline{r}_{ℓ} are the volume and position associated with a neighboring grid point.

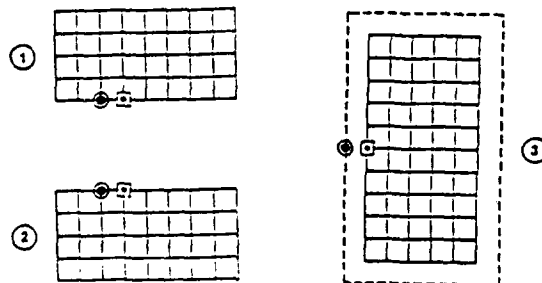
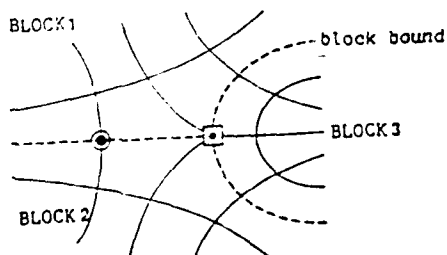
For example, the Cartesian coordinates of the special point illustrated below by the solid circle would be set as the average of those at all of the circled points:



Here the input statement is

\$INPUT ITEM = "AVERAGE", BLOCK = __, POINT = __, __, __ \$

Alternatively, it is possible to construct a local coordinate system at a special point by an input statement with ITEM = "FIELD" (including BLOCK and POINT to identify the special point), to designate the special point as a "FIELD" point to be generated by the grid generation system, followed by one or more input statements with ITEM="IMAGE" (including BLOCK, POINT, IBLOCK, and IPOINT). Here with ITEM="IMAGE", the Cartesian coordinates of the point identified by IBLOCK and IPOINT (the image point) will be kept equal to those of a point identified by BLOCK and POINT (the object point). Thus in the illustration below, the special point is that with the open square, and a local coordinate system is set up according to the dotted lines by making the open-circle point the image of the solid-circle point:



The input here is as follows:

```
$INPUT ITEM = "CUT", BLOCK = 3, START = 1,6, END = 1,11,  
    IBLOCK = 1, ISTART = 4,1, IEND = 9,1 $  
$INPUT ITEM = "CUT", BLOCK = 3, START = 1,6, END = 1,1,  
    IBLOCK = 2, ISTART = 4,5, IEND = 9,5 $  
$INPUT ITEM = "CUT", BLOCK = 1, START = 1,1, END = 4,1,  
    IBLOCK = 2, ISTART = 1,5, IEND = 4,5 $  
$INPUT ITEM = "FIELD", BLOCK = 3, POINT = 1,6 $  
$INPUT ITEM = "IMAGE", IBLOCK = 3, IPOINT = 0,6,  
    BLOCK = 1, POINT = 3,1 $
```

The statement with ITEM = "FIELD" changes the classification of the special point (1,6) in block 3 to "FIELD". The last statement is necessary to establish the local coordinate system at the special point by changing the object point of the image point (0,6) on the surrounding layer to the left of the special point in block 3 from point (4,2) in block 1, as set by the first "CUT" statement, to point (3,1) in block 1. The points (4,1) in block 1 and (4,5) in block 2 are both image points having the special point as their common object point. The choice of block 3 as the block in which the special point is designated a field point is arbitrary, of course. There are several equivalent setups here, all of which yield the same grid in the physical region.

13. Smoothing the Grid or Control Functions

Values of the Cartesian coordinates or control functions at points within a section can be smoothed by an input statement with ITEM="SMOOTH", the section again being identified by BLOCK, START, and END.

The smoothing is done only in the curvilinear directions entered in SMODIR (the default is for all three directions). The Cartesian components, or the control functions, to be smoothed are specified by entries in SMOCOM (the default is all three components). If FUN = "BOUND", the smoothing is done on the boundary values before the code interpolates for the algebraic grid within the block, or is done on the algebraic grid if FUN="POINTS" (the default). If FUN="CONTROL", the control functions are smoothed. Each control function indicated by SMOCOM is smoothed only in the other two directions, i.e., P_1 is smoothed in the 2 and 3 directions, unless SMODIR indicates smoothing in only a single direction. All the blocks can be treated by a single input statement with ALL="YES" included instead of BLOCK, START, and END.

14. Algebraic Grid

The code generates an algebraic grid in the interior of all blocks by transfinite interpolation to serve as the initial solution for the elliptic grid generation system. The form of this interpolation is controlled by an input statement with ITEM="INITIAL" including all of the interpolation parameters discussed in Section I-C6, except that the spacing for Hermite interpolation is automatically interpolated from the sides. The same interpolation form can be done for all blocks by one statement with ALL="YES", or can be done separately for each block by a series of input statements including BLOCK to specify the block number. It is possible to turn off this interpolation by including PRODIR="NONE"

if the initial grid is being read in. It is also possible to turn off the elliptic generation system by simply omitting ITMAX, and thus to take this algebraic grid as the final result.

15. Control Functions

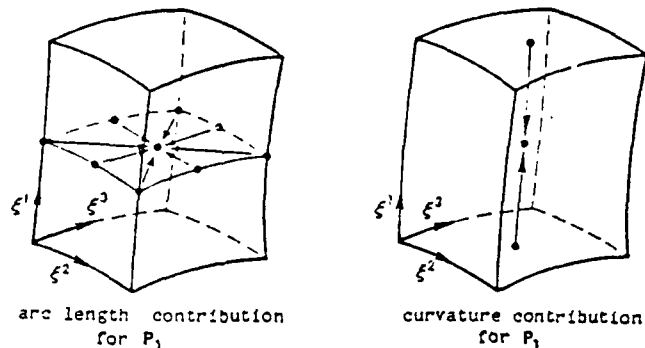
The mode of determination of the control functions by the code is controlled by the variable CONTYP which need appear only once, anywhere in the input. If CONCUR = "AVERAGE" is included, the control functions on all cuts will be averaged across the cut. This is not generally recommended. One of five modes is specified by CONTYP.

A value of "NONE" for CONTYP gives a Laplace grid, i.e., with the control functions set to zero everywhere. This is the smoothest possible grid, but may not have a suitable concentration of lines in that the boundary point distributions have little effect in the field.

With CONTYP="INITIAL" the control functions are evaluated directly from the initial algebraic grid by substitution of the initial grid points into the elliptic grid generation system. This would actually be trivial, in that the algebraic grid would be returned unchanged by the generation system in one iteration, except that these control functions are smoothed, before being used. These control functions will reflect the same relative grid line distribution from the initial algebraic grid but with more smoothness and less skewness. This is actually the preferred mode, especially in more complicated block configurations, and hence is the default.

The modes "ORTHO" and "ORTHO2" also determine the control functions from the initial algebraic grid but with all terms arising from nonorthogonality, i.e., the off-diagonal metric elements, discarded for the former. With "ORTHO2" terms arising from nonorthogonality of ξ^j and ξ^k lines when the control function P_i (i,j,k cyclic) is evaluated are retained. There is really little difference between these two modes in most cases. These two modes will produce a grid that is somewhat more orthogonal than the initial algebraic grid, but may not always be smooth and may have skewness at some boundaries.

The mode "RADIUS" evaluates the P_i control function by interpolating the contribution from the boundary point arc length distribution between the four sides (two in 2D) of the block on which the ξ^i coordinate varies, and interpolating the curvature contribution between the other two sides.



This provides for the retention of the boundary point distribution in the field, but is not generally usable with "OUT" points in the block. Such a block, however, can always be broken into several smaller blocks using the sub-block feature (Section I-C16).

The mode "SIDES" operates in a similar manner to "RADIUS" except that the curvature contribution is interpolated together with the arc length contribution. This mode is not effective if the curvature varies widely between the boundaries.

Although the specification of the control function mode is normally made once for the entire system, it is possible to make the specification separately for sections of grid points by an input statement with ITEM="CONTROL", using BLOCK, START, and END to identify the section. In this case the evaluation and interpolation specified by CONTYP occurs within the section, using the section boundaries as if they were block boundaries. Point distributions on all sides of the section must have been established before this operation is invoked. (This procedure is not generally recommended.)

Finally, the control functions can be evaluated iteratively by the grid generation system so that the grid is orthogonal to a boundary section, with a specified spacing off that boundary, by an input statement with ITEM="ORTHOG", using BLOCK, START, and END to identify the section, as has been discussed in Section I-C9. Only points not already classified "FIX" will be affected. The control functions will be updated at intervals specified by an integer entry in CONUPI during the iteration of the grid generation system. In this case CONTYP serves only to set the initial control functions and should generally be left to the default.

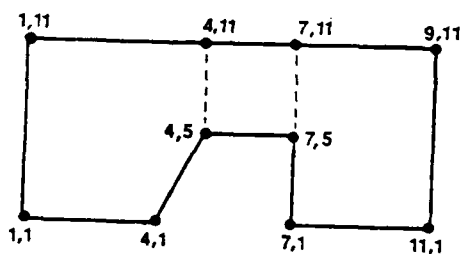
16. Sub-Block Structure

In some more complicated configurations, particularly those with physical boundaries in the interior of a block or on only a portion of a block side, it is advantageous to set up a sub-block structure for the interpolation for the algebraic grid and/or for the evaluation of the control functions. This proceeds as follows:

Once a block has been specified (Section I-C2), a series of input statements with ITEM="SUB", including BLOCK, START, and END, can be given to define sub-blocks within the block. If this structure is used, the sub-blocks must not overlap and must completely fill the block (except for any sections of "OUT" points).

If no points are set on a sub-block boundary, values will automatically be interpolated on that boundary if possible as described in Section I-C6.

An example follows:



```
$ITEM = "BLOCK", SIZE = 9,11 $
```

```
$ITEM = "SUB", START = 1,1, END = 4,11 $
```

```
$ITEM = "SUB", START = 4,5, END = 7,11 $
```

```
$ITEM = "SUB", START = 7,1, END = 9,11 $
```

Here, if values are read or interpolated for START=4,5, END=4,11 and for START=7,5, END=7,11 the interpolation for the algebraic grid will be done within the three sub-blocks.

The sub-block structure affects only the interpolation for the algebraic grid and the evaluation of the control functions, and is irrelevant to all other features.

17. Storage

The final grid can be stored on a file by an input statement with ITEM="STORE", with the file number as the integer FILE. The total number of blocks (the integer BMAX), and the integer array CMAX containing the size of each block are written on the file first, unformatted as

```
BMAX,((CMAX(CI,B), CI=1,3, B=1,BMAX)
```

Then the arrays TYPE, IMAGE, and R are written unformatted by

```
DO    B = 1, BMAX
DO    C3 = IS, CMAX(3,B) + NS
DO    C2 = IS, CMAX(2,B) + NS
DO    C1 = IS, CMAX(1,B) + NS

WRITE(__) TYPE(B,C1,C2,C3)
WRITE(__) (IMAGE(CI,B,C1,C2,C3), CI=0,3)
WRITE(__) (R(CI,B,C1,C2,C3), CI=1,3)
```

Here IS=0 and NS=1 if the code has been compiled for one surrounding layer of points on each block, or IS=-1 and NS=2 for two layers. These parameters are set in PARAMETER statements by global edits. If OUTER="NO" is included, then the surrounding layer of points outside

each block will not be put on the file, i.e., the DO limits are 1 and CMAX. In this case, the arrays TYPE and IMAGE are not written on the file. The inclusion of BMAX and CMAX on the output file is suppressed if OUTER="BOUND" is included. In this case, only the coordinate array R is put on the file, and without any surrounding layers.

If FILE is equal to "RESTART", instead of a number, a restart file will be written on file 7 from which the iteration can be continued. If both a restart file and storage of the grid as above are desired, two of these input statements must be given. File 7 cannot be used as a storage file for the grid.

All or sections of all or certain blocks can be stored on a separate file (8) for plotting as discussed in Section I-D.

19. Iterative Solution Parameters

The maximum number of iterations allowed for the elliptic grid generation system is specified by an integer entry in ITMAX. Omission of ITMAX will cause the initial algebraic grid to be output as the final grid, bypassing the elliptic grid generation. The convergence tolerance for the iteration is specified by a real entry in TOL. This convergence tolerance is relative to the maximum extent of the physical region in any of the three Cartesian directions. If TOL is omitted, the total number of iterations indicated by ITMAX will be done. The acceleration parameter for the iteration is a real entry (between 0 and 2) in ACCPAR. This value should normally be around 1.3, and should be decreased for small fields or strong concentration of lines. If ACCPAR="OPTIMUM" (the

default), a variable acceleration parameter field will be set up and this choice is recommended. These parameters need appear only once, anywhere in the input.

19. Jacobian Check

After the initial algebraic grid has been generated, the code checks for points with zero Jacobian, i.e., zero cell volume (usually indicating failure to input all necessary boundary or interface points), or a locally left-handed coordinate system. A left-handed system may be an indication of a twisted grid resulting from an inconsistent ordering of points on some boundary segment, or may be simply a case of the algebraic grid overlapping a boundary. In the latter case the elliptic generation system may be able to rectify the grid. The action taken by the code upon discovery of either of these phenomena is controlled by the value input for the integer CHECK. A value of "YES" causes the code to simply stop before starting the iteration in either case. A value of "NO" (the default) causes a stop in the case of a zero Jacobian, but allows the generation to continue with the left-handed system. In any case of a stop, the algebraic grid is generated and the output proceeds as specified. The parameter CHECK need appear only once, anywhere in the input.

20. Derivative Difference Forms

The difference expression for the first derivatives is set by the integer DFIRST. The value "CENTRAL" gives two-point central differences:

$$\odot \quad \times \quad \odot$$

while "ONESIDE" gives two-point one-sided differences, the direction of which depends on the sign of the control function:

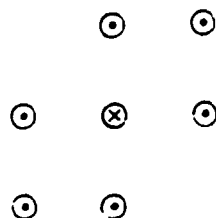
$$\odot \quad \otimes$$

The value "VARIABLE" gives central differences with control functions that do not exceed 2.0 in magnitude, with a weighted average between central and one-sided otherwise. The average changes inversely with the control function to fully one-sided for very large values. The default is "VARIABLE".

The difference expression for the cross derivatives is set by the integer DCROSS. The value "CENTRAL" (the default) gives four-point symmetric differences:

$$\begin{array}{cc} \odot & \odot \\ & \times \\ \odot & \odot \end{array}$$

while "ONESIDE" gives a diagonal form depending on the sign of the off-diagonal metric elements.



These parameters need appear only once, anywhere in the input.

21. Block Storage

The code is set up to treat one block at a time in order to save storage. This is accomplished in one of two ways, controlled by the value given for the quantity KSTORE which need appear only once but must be included on the first input statement. If KSTORE="SSD" (the default), each block is kept on a separate disk file to be accessed as needed. These files are numbered BASE + block number, where the integer BASE is set to 30 in a PARAMETER statement but must be changed by a global edit. All of these files should be assigned to a solid-state disk in the job control language, if one is available, otherwise considerable time and possible expense will be spent in IO.

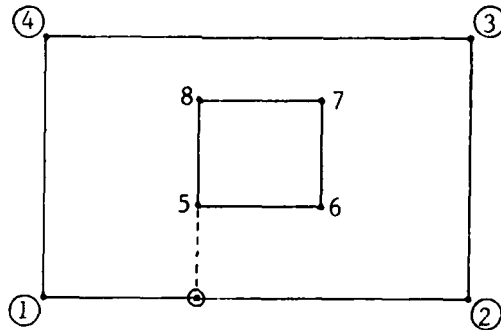
The other alternative is to keep all blocks in core via KSTORE="CORE".

22. Restart

The code can be restarted from the algebraic grid by including RESTART="INITIAL" on any input statement, or the iteration can be continued by including RESTART = "ITERATE".

23. Numbered Points

It is possible to reference a point by a single number instead of giving the three indices (ξ^1, ξ^2, ξ^3) of the point. This feature, and the companion feature in the front-end boundary code, allows the number of points on a segment to be changed where the segment is generated in the front-end without requiring changes in the input to the grid code. These features also greatly simplify the set up of the block structure for complex configurations. If a negative value is given for any of the three entries of START, END, ISTART or IEND, the corresponding index is taken from the table of points according to the point number indicated by the magnitude of the negative value. This feature allows indices to be taken from different points. Thus the circled point in the illustration below



can be referenced by START=-5,-1 (or equivalently by -8,-2). Positive and negative values can also be mixed, with the former indicating indices directly.

When only a single positive value is given for START, END, ISTART, IEND or SIZE, that value is taken as the point number, and the three indices (ξ^1, ξ^2, ξ^3) for the point are obtained from a table of points that has been set up by a series of input statements with ITEM="POINT". These statements can set the point indices directly by

\$INPUT ITEM = "POINT", POINT = point number, LOCAT = ξ^1, ξ^2, ξ^3 \$

However, it is also possible to utilize information from the front-end surface code, stored there with ITEM="COMBINE" when boundary segments are combined onto a single file (cf. Vol. II, Section I-E21). This usage is as follows:

When segments are combined onto a single file by the ITEM="COMBINE" operation in the front-end code, a table of contents is written at the beginning of the file if CONTENT="YES" is included. This table includes the COREOUT number and the dimensions of each segment on the file. (Each segment in the combination must have been stored in core using COREOUT when generated, since the COREOUT number becomes the segment number.)

This entire file of boundary segments generated by the front-end code can be read by the grid code using the input statement

\$INPUT ITEM = "FILE", FILE = file number, ALL = "YES" \$

This file can contain segments from several blocks, but must include all the segments associated with those blocks.

A block is then introduced by an input statement with ITEM="BLOCK". Here SIZE can be included with three values as originally, or can be omitted to be specified later as discussed below in terms of a point

number. (If all the segments are associated with a single file, the file can alternatively be read by including the file number as FILE, together with the associated file parameters, cf. Section I-C3, on the ITEM="BLOCK" statement.)

After the block has been introduced, all the numbered points associated with segments on the file can be set up by a series of input statements of the form

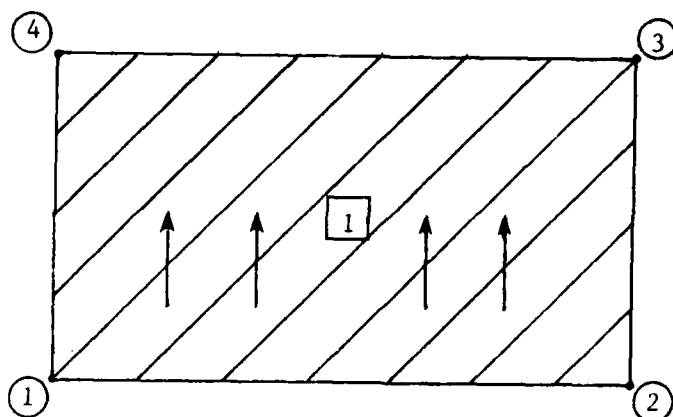
```
$INPUT ITEM = "POINT", POINT = point number,  
OPOINT = other point number, SEGMENT = segment number,  
DIRECT = __, NDEX = __ $
```

Here POINT gives the number of the point being set, OPOINT is the number of another corner point on the segment identified by SEGMENT, DIRECT is +1, +2, or +3, indicating which index (ξ^1 , ξ^2 , or ξ^3) changes from OPOINT to POINT and is positive(negative) if that index increases(decreases), and NDEX is 2 if the point progression from OPOINT to POINT corresponds to the slower running dimension on the segment or is omitted otherwise. One point number on a closed circuit must be set up directly with the statement

```
INPUT ITEM = "POINT", POINT = point number, LOCAT =  $\xi^1$ ,  $\xi^2$ ,  $\xi^3$  $
```

before the others on the circuit follow.

As an example, if the segment #1 below was generated by the front-end code with the faster running point progression as indicated by the arrows,



the four numbered points can be set as follows:

```
$ INPUT ITEM = "POINT", POINT = 1, LOCAT = 1,1,1 $
$ INPUT ITEM = "POINT", POINT = 2, OPOINT = 1, SEGMENT = 1,
    DIRECT = 1, NDEX = 2 $
$ INPUT ITEM = "POINT", POINT = 3, OPOINT = 2, SEGMENT = 1,
    DIRECT = 2, NDEX = 1 $
$ INPUT ITEM = "POINT", POINT = 4, OPOINT = 3, SEGMENT = 1,
    DIRECT = -1, NDEX = 2 $
```

Note that this setup sequence is not unique, e.g., point 4 could have been set from #1, but the result is unique. Also, there is no required order, and any of the points could have been set directly first. If this segment has dimensions (5,4) from the front-end code, the point indices here will be as follows:

<u>point</u>	<u>ξ^1</u>	<u>ξ^2</u>	<u>ξ^3</u>
1	1	1	1
2	4	1	1
3	4	5	1
4	1	5	1

A point can also be set from another point without reference to stored segments by including the number of points joining the point to the other point, instead of SEGMENT and NDEX. Thus, if point #1 is at (1,3,2) and point #2 is set by

```
$INPUT ITEM = "POINT", POINT = 2, OPOINT = 1, POINTS = 5, DIRECT = 2 $
```

the indices of point #2 will be (1,7,2).

It is also possible to create a group of new points from a group of other points by including ALL="YES", and either POINTS or SEGMENT and NDEX. In this case a set of new points will be created from all the presently created points. The new points will be numbered as the other point numbers, plus the value given as POINT. This feature is normally used to create points on an opposite side of a block having the same segment configuration, e.g. for a body of revolution.

After all the points in the block have been set, all the boundary segments associated with that block are put in place by a succession of input statements of the form

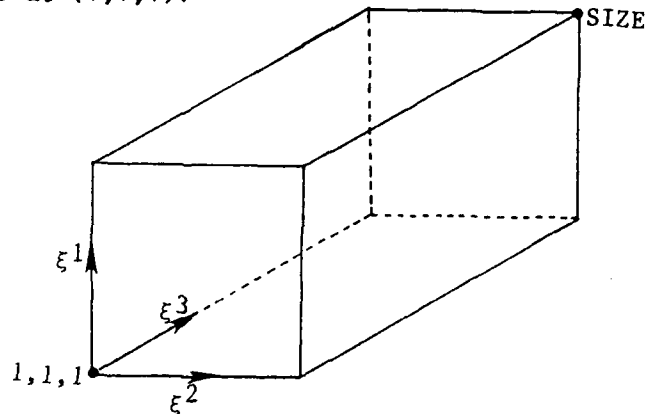
```
$INPUT ITEM = "SEGMENT", SEGMENT = segment number,  
START = point number, END = point number $
```

Here CLASS="FIX" can be included if appropriate, as can IPRINT (cf. Section I-C3). The segment number given on this statement corresponds to the value of COREOUT used in connection with this segment in the input for the front-end code, as has been noted. The order in which these statements appear does not have to conform to the order in which the segments were written on the file.

Finally, the size of the block can be set by the input statement

```
$INPUT ITEM = "SIZE", SIZE = point number $
```

where the point number given is that of the block corner most remote from the point at (1,1,1):



This procedure is followed for each block associated with the file of segments. Other files with segments from other groups of blocks can then follow.

The same point number can, and should, be used in several blocks for a common point in physical space.

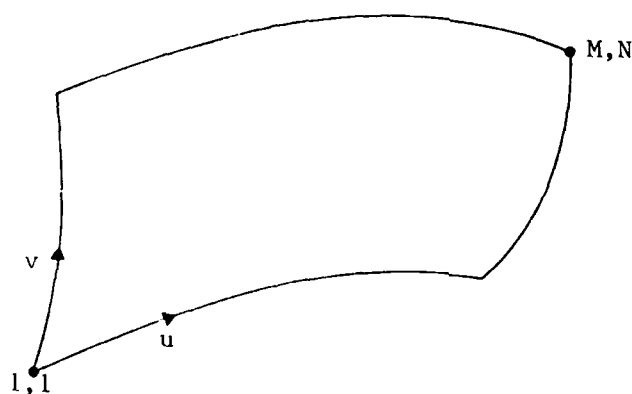
24. 2D Grid on Curved Surface

The code can generate a 2D grid on a curved surface as follows:

The 2D array of points defining the surface is first read in by an input statement with `ITEM="SURFACE"`, with the dimensions of the surface given by including the array `SIZE(2)`. The three Cartesian coordinates of the points defining the surface are read from a file if the file number is included as `FILE`, or from the namelist if `VALUES` is included (cf.

Section I-C3). The reading is done as is described in Section I-C3, but without BLOCK, START and END. This surface can, of course, have been generated by the front-end code or some other code.

This surface is splined and the code then functions in a 2D mode, but in terms of the two spline coordinates, i.e., surface parametric coordinates. These two parametric coordinates (u, v in the figure below) are simply the indices of the curved surface array, i.e., they vary from 1 to the dimensions (M, N below) of the surface in each of the two families of lines forming the surface.



The two coordinate values that are read for points on boundary segments on the grid to be generated must be values of these parametric coordinates. These values can conveniently be generated by the front-end code using the ITEM="CORPAR" operation to convert Cartesian coordinate values for points on a curve to parametric coordinates after the curve has been generated on the curved surface using the SURFACE="CURVED" mode in the front-end code (cf. Vol. II, Section I-B8). Thus the curved surface can be generated by the front-end code and stored on file for reading into the grid code (via ITEM="SURFACE

described below). Next in the front-end code, this surface can be splined and the boundary edges of the grid can be generated on this surface using the SURFACE="CURVED" mode there. The Cartesian coordinates of the points on these edges then can be converted to parametric coordinates (by ITEM="CORPAR" there) and stored for reading into the grid code as the boundary segments (via ITEM="FILE", etc. in the grid code).

The elliptic generation system is that with the parametric coordinates as the dependent variables, taking full account of the curvature of the surface. All features of the code, e.g., block structure, boundary orthogonality, etc., apply in this mode as well. After generation is complete, the parametric coordinates are converted back to Cartesian coordinates thus providing the 2D grid on the curved surface. On the output for plotting, the curved surface is automatically output as block #1, and all of the actual block numbers of the grid are incremented by one on this output.

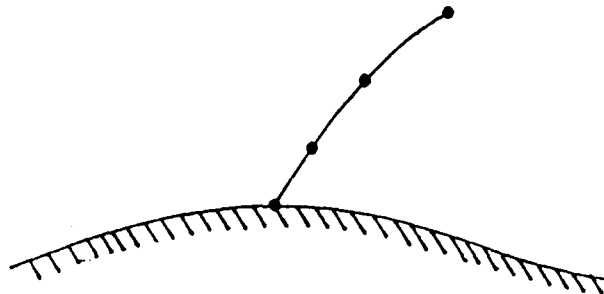
25. Continuity Check

It is possible in rare instances for cuts to be set up in such a way that a point on a block interface is imaged to a field point in one block, while an adjacent point on the interface is imaged to a non-field point in another block. Such a situation can produce a local loss of derivative continuity and can often be avoided by reversing the designation of the object and image blocks at the interface. This situation

can be checked for by including CONTIN="YES" on some input statement. Since this checking can require a large number of accesses to the disk storage, it can be expensive and therefore the default is not to check.

26. Zero-Curvature Extrapolation at Boundary

Neumann boundary conditions based on zero curvature of lines intersecting the boundary instead of orthogonality can also be applied:



The usage is the same as described for ITEM="NEUMANN", but with ITEM="EXTRAP". The "NEUMAN1", "NEUMAN2", and "NEUMAN3" statements can also follow an "EXTRAP" statement to restrict the movement.

D. NAMELIST/OUTPUT/

The printed output and file storage for plotting is controlled by one or more successive reads of a NAMELIST called OUTPUT, with the type of output being specified by ITEM="___" on each such input statement. This portion of the input is terminated by an input statement with ITEM="END".

As in the NAMELIST/INPUT/, numbered points can be used as single entries in START and END (cf. Section I-C23).

On all the OUTPUT operations, the inclusion of PART="SIDES" causes all sides of a block to be output, and "EDGES" produces all edges.

1. Printout

With ITEM="INITIAL", the initial algebraic grid will be printed, while "PRINT" prints the final grid. In each of these cases, the sections to be printed are identified by BLOCK, START, and END. Any number of these input statements can be used to print several portions of the field. The entire field will be printed if ALL="YES" is included instead of BLOCK, START, and END. The order in which the points are printed is controlled by ORDER, and the order in which the Cartesian components are printed for each point is controlled by RORDER, in the same manner that has been described for reading the boundary points (Section I-C3). Again zero entries here cause corresponding omissions from the printout.

2. Plot File

All or sections of all or certain blocks can be written on file 8 for plotting in the same manner with ITEM="PLOT". The points are written one point (three Cartesian coordinates) to a line in a format set by FORM="UNFORM" for unformatted, "E" for 3E20.8, or "LIST" for list-directed. In contrast to the print-out, all three coordinates are written on the file in any case.

3. Error Norm

The printing of the error norm at each iteration is activated by an input statement with ITEM="ERROR". If BLKERR="YES" is included, then the error norms in each block will be printed. Otherwise, only a single norm over all the blocks is printed. If the ITEM="ERROR" input statement is used, it must be the first of the namelist OUTPUT statements.

E. ERROR MESSAGES

ALL POINTS ON SECTION ARE FIXED

This warning occurs when all points on a NEUMANN, EXTRAP, or ORTHOG section are FIX points. Since FIX points are not re-classified, the NEUMANN, EXTRAP, or ORTHOG designation here is totally ineffective.

BAD SEGMENT LIST ON FILE

This occurs when all segments for a block are being read from a file at once (Section I-C3), but the number of segments recorded on the file is 0. Check to see that the file was generated by the boundary code on ITEM="COMBINE" with CONTENT="YES" (Section I-E21 of Vol. II).

BAD VALUE FOR ---

This indicates that an unrecognizable value has been given for the quantity indicated, perhaps simply because of misspelling. The acceptable values for all such quantities are listed below:

NS	:	1,2
KSTORE	:	"FILE", "CORE"
CLASS	:	"FIX", "FIELD", "OUT"
DFIRST	:	"CENTRAL", "ONESIDE", "VARIABLE"
DCROSS	:	"CENTRAL", "ONESIDE"
CHECK	:	"YES", "NO", "PLOT"
RESTART	:	"INITIAL", "ITERATE", "NO"
PROTYP	:	"FACES", "EDGES", "CORNERS"
CONFTYP	:	"RADIUS", "INITIAL", "ORTHO", "ORTHO2", "NONE", "SIDES"
INTERP	:	"YES", "NO"

TRIAD	:	"YES", "NO"
IPRINT	:	"YES", "NO"
INTORT	:	"YES", "NO"
ITMAX	:	Non-negative integer
CONUPI	:	Non-negative integer
ACCPAR	:	"OPTIMUM" or Real number between 0 and 2
TOL	:	Non-negative real number
SPAVAL	:	Non-negative real number
WEIGHT	:	"YES", "NO"
REWIND	:	"YES", "NO"
ALL	:	"YES", "NO"
OUTER	:	"YES", "NO"
NEW	:	"YES", "NO"
SMOCON	:	"YES", "NO"
FUN	:	"POINTS", "CONTROL"
SEGMENT	:	Positive integer
DIRECT	:	I1, I2, I3, "UP", "DOWN"
NEDX	:	1, 2
PROFOR	:	"LAGRANGE", "HERMITE", "HERMITE1", "HERMITE2"
BLEND	:	"LINEAR", "ARC"
ORDER	:	1, 2, 3
RORDER	:	1, 2, 3
PRODIR	:	1, 2, 3, "NONE"
SMODIR	:	1, 2, 3
SMOCOM	:	1, 2, 3
VALOUT	:	Positive integer

MATH : "SUM", "SUM-1", "DIF", "DIF-1", "PRODUCT"
ITEM (On OUTPUT): "INITIAL", "PRINT", "PLOT", "ERROR", "END"
CONFAC : positive real
PART : "SIDES", "EDGES", "ALL"
BLKERR : "YES", "NO"

BLOCK NUMBER EXCEEDS TOTAL USED

The value gives for BLOCK is greater than the total number of blocks that have been set up. A block cannot be addressed until it is set up by an ITEM="BLOCK" statement. Blocks are numbered consecutively as they are set up (Section I-C2).

<BLOCK & POINT> ARE REQUIRED

This occurs when a point is being classified FIELD, IMAGE, or AVERAGE, but either the block or the point is not designated. In the case of an IMAGE point this indicated that no object point was designated.

BOUNDARY VALUES MUST BE SPECIFIED

This occurs when the code is unable to set all the boundary values on the sides of a section for interpolation, either through reading the values or through interpolation on the sides. The minimum required is the specification of values at all the corner points.

COORDINATE SYSTEM IS NOT RIGHT-HANDED

This indicates that the unit vectors in the three curvilinear coordinate directions do not form a right-handed system (Section I-C19). The probable cause is either a twisted grid because the points on some boundary segment have been input in the wrong direction or an overlapped grid resulting from an overlap of a boundary segment in the transfinite interpolation. The former is a user error and must be corrected. The latter is not a user error and may be resolved by the elliptic grid generator. (A later check is made after the elliptic grid is generated.) The option CONTYP="INITIAL" for the control functions cannot be used with an overlapped algebraic grid, however. In any case the algebraic grid should be plotted and examined at the location indicated. A non-right-handed final grid should be plotted and examined at the location indicated. A non-right-handed final grid will not be usable in a PDE solver. The code can be made to stop at this message by including CHECK="YES".

CUTS SHOULD NORMALLY BE LAST

The ITEM="CUT" statement causes the Cartesian coordinates at corresponding points on pairs of interfaces to be kept equal. In order for this correspondence to be set up properly, it is normally necessary that all points on the cut section that are to be classified FIX, ORTHO, NEUMANN, EXTRAP, or OUT receive that classification before the cut is set up (Section I-C7). Therefore all the CUT statements should be grouped at the very end of the INPUT statements. (CUT statements within a single block can be put at the end of the statements for that block.)

<DIMR> NOT SET FOR CORE STORAGE
<DIMR> SHOULD BE SET TO 1 FOR FILE STORAGE

DIMR is the total number of points allowed in all blocks when all blocks are kept in core (Section I-C21). Since this parameter is naturally set to 1 when blocks are kept on separate files instead of in core, the code checks to see if the value has been changed from 1 when core storage is called for. Similarly, in order not to waste storage, a check is made to see that DIMR is set to 1 when file storage is called for.

<DIMS> MUST NOT BE LESS THAN <DIMC>

This occurs in the curved surface mode (Section I-C24) when the maximum size of the surface allowed, DIMC, exceeds the dimension DIMS. Increase DIMS globally unless DIMC is unnecessarily large.

15*DIMT MUST NOT BE LESS THAN 12*DIML**2

This requirement occurs because of some equivalence used to conserve storage. The solution is normally to increase DIMT in a global edit, unless DIML is unnecessarily large.

<DIRECT> IS REQUIRED

This occurs when the indices for a numbered point are being set (Section I-C23), but no direction on the grid is given.

<END> IS OUT OF BLOCK

Analogous to <START> IS OUT OF BLOCK.

<END> POINT NOT SET

Analogous to <START> POINT NOT SET.

EXTRAP BOUNDARY DOES NOT CONVERGE

See the corresponding NEUMANN statement.

<FIELD> POINT --- ADJACENT TO AN <OUT> POINT

This indicates that one of the 26 points (8 in 2D) adjacent to a FIELD point is an OUT point. The probable cause is that the FIELD point is on a block boundary that has not been made a cut or a FIX, NEUMANN, EXTRAP, REFLECT, or ORTHOG section. Since all points on the block are defaulted to FIELD, it is necessary to make some classification of all boundary points.

<FIELD> POINT --- IS ADJACENT TO A POINT WITH NO VALUE

This indicates that one of the 26 points (8 in 2D) adjacent to the FIELD point has no assigned value for its Cartesian coordinates. The probable cause is a failure to read in some boundary points.

<FILE> IS REQUIRED

This occurs when no file is designated for the final grid (Section I-C17). If no such storage is intended, the ITEM="STORE" statement should not be used.

FORBIDDEN FILE NUMBER

File numbers 1-10 cannot be used to read points or to store the grid.

<IEND> IS OUT OF BLOCK

Analogous to <START> IS OUT OF BLOCK.

<IEND> POINT NOT SET

ANALOGOUS TO <START> POINT NOT SET.

IMAGE & OBJECT EDGES IN --- DIRECTION DO NOT CORRESPOND

This occurs when the image (ISTART, IEND) and object (START,END) section of a cut, (Section I-C7) are not the same size. This is influenced by ORDER as well, i.e., $IABS(IEND(I) - ISTART(I))$ must equal $IABS(END(ORDER(I)) - START(ORDER(I)))$ for $I=1,2,3$.

<IMAGE> POINT --- IS ADJACENT TO A POINT WITH NO VALUE

See the corresponding <FIELD> statement.

INITIAL GRID POINT MUST COME FIRST IN OUTPUT NAMELIST

When a print of the initial grid is called for, (Section I-D1) the ITEM="INITIAL" statements must precede all of the "PRINT" and "PLOT" statements.

<IPOINT> IS REQUIRED

This occurs when an IMAGE point is being set, but no image point is designated.

<ISTART> IS OUT OF BLOCK

Analogous to <START> IS OUT OF BLOCK.

<ISTART> POINT NOT SET

Analogous to <START> POINT NOT SET.

<ITERMS> IS REQUIRED

This occurs when an integer value is being set by a calculation (Section I-C1).

JACOBIAN IS ZERO

This indicates that a zero cell volume has been found (Section I-C19). The probable cause is that all points in a region have the same value through an input error.

<KSTORE> CANNOT APPEAR AFTER THE FIRST INPUT STATEMENT

Since KSTORE sets the mode of block storage (Section I-C21), i.e., all in core or separately on fields, it must be given on the first input statement, which is most appropriately the ITEM="INITIAL" statement.

LOCATION IS OUT OF BLOCK

This occurs when the indices of a numbered point are being set (Section I-C23), but the indices given in LOCAT are out of the block.

NAMelist ITEM NOT RECOGNIZABLE

This occurs when a value given for ITEM is not one of the operations provided. Check for misspelling.

NEGATIVE OR ZERO RESULT

This indicated that a calculated integer value to be stored (Section I-C1) is not positive.

NEUMANN BOUNDARY ARRAY TOO SMALL

This indicates that there are too many NEUMANN or EXTRAP points in a block for the dimensions set (Section I-C9, C26). Increase the dimension parameter globally. The suggested value is adequate only for the instance noted.

NEUMANN BOUNDARY DOES NOT CONVERGE

This indicates that the Newton iteration for a NEUMANN point has not converged (Section I-C9, C26). The probable cause is the restriction preventing such points from leaving the section, i.e., the points lock at the section edge, or a storage configuration. The grid should be plotted and examined in the area indicated.

<NEUMANN> POINT --- IS ADJACENT TO AN <OUT> POINT

See the corresponding <ORTHOG> statement.

<NEUMANN> SECTION CANNOT BE A ---

This occurs when a NEUMANN or EXTRAP section has been specified by START and END to be volume in 3D or a surface in 2D. One pair of corresponding entries in START and END must be equal in 3D, and two pairs must be equal in 2D, so that the section is a surface in 3D and a line in 2D (Section I-C9, C26).

NO ORTHOG CONVERGENCE

This indicates that convergence was not obtained in the Newton iteration for the spacing distribution with ORTHOG points in subroutine CONSETE (Section I-C10). This could be the result a strange configuration, perhaps one in error through bad boundary data.

NO RAD CONVERGENCE

This occurs when the Newton iteration for the radius of curvature for use with the CONTYP="RADIUS" control function option fails to converge (Section I-C15). The probable cause is a storage configuration, perhaps through input error. The algebraic grid should be plotted and examined.

NO SOURCE FOR SURFACE

This occurs in the curved surface mode (Section I-C2⁴) when no points are given for the surface, either by indicating a read from a file by including FILE or directly from the NAMELIST by including VALUES.

NO STORED VALUE FOR ---

This occurs when a negative integer is given for a component of START, END, ISTART, IEND, or SIZE and no value has been stored, by a prior ITEM="SETNUM" (Section I-C1) in the location indicated by the magnitude of the negative value.

NO STORED VALUE IN LOCATION

This occurs when a value is being calculated to be stored (Section I-C1) and one of the terms of the calculation indicated as by previously stored is, in fact, not stored.

<NS> CANNOT BE 0

NS is the number of surrounding layers. Since the code must have one surrounding layer, <NS> must be 1 or 2. Use 2 only if it is desired to have two surrounding layers output for use in a PDE code. The correct way to suppress the output of the surrounding layer is to use OUTER="NO" on the input, not to set NS=0.

OBJECT POINT IS IMAGE POINT

This occurs when the object of an image point is itself an image point. If the image point in question is actually needed, the code will stop. The probable cause is an improper cut setup if the code stops. This is not a problem if the code continues.

OBJECT POINT WAS OUT POINT

This occurs when the object of an image point was found to be an OUT point. In this case the image point is changed to an OUT point also. If the image point was needed, the code stops. Otherwise this is not a problem. If the code does stop, the probable cause is a bad cut setup.

<OPOINT> IS REQUIRED

This occurs when the indices for a numbered point are being set relationally from another point (Section I-C23), but the other point is not given.

<ORDER> MUST CONTAIN ONLY 1 & 2

This occurs in the curved surface mode (Section I-C24) when a value 3 is given for ORDER(1) or ORDER(2). Since the surface has only two dimensions, only 1 and 2 are acceptable values for ORDER here. ORDER(1) is the direction on the surface that is read in the inner loop.

<ORTHOG> POINT --- IS ADJACENT TO AN <OUT> POINT

This indicates that one of the necessary points adjacent to the ORTHOG point is an OUT point. The probable cause is that the ORTHOG point is on a block edge that is not part of a cut interface.

<ORTHOG> SECTION CANNOT BE A ---

See corresponding <NEUMANN> statement.

ORTHOGONAL BOUNDARY ARRAY TOO SMALL

Analogous to <NEUMANN BOUNDARY ARRAY TOO SMALL.

OTHER POINT HAS NOT BEEN SET

This occurs when the indices of a numbered point is being set relationally from another numbered point (Section I-C23), but no indices have been set for the other numbered point.

<POINT> IS REQUIRED

This occurs when the indices of a numbered point are being set (Section I-C23), but the point number is not given.

<POINTS> IS REQUIRED

This occurs when the indices of a numbered point are being set relationally from another point (Section I-C23), but no increment is given for the indices, either directly by including POINTS or indirectly by including SEGMENT.

POINT NUMBER EXCEEDS INCREMENT

This occurs when a new group of numbered points is created by incrementing all existing point numbers in a block (Section I-C23) and the increment given is not adequate to place the new point numbers beyond the range of the present numbers. Use a larger increment.

POSSIBLE LOSS OF CONTINUITY

See Section I-C25.

<REFLECT> POINT --- IS ADJACENT TO AN <OUT> POINT

See the corresponding <ORTHOG> statement.

<REFLECT> SECTION CANNOT BE A ---

See corresponding <NEUMANN> statement.

SEGMENT DOES NOT MATCH <START,END>

This occurs when points on a segment are being placed (Section I-C23) after all segments for a block have been read from a file at one time, but the segment does not find the placement indicated by START and END.

<SEGMENT> IS REQUIRED

This occurs when points on a segment are being placed (Section I-C23) after all segments for a block have been read from a file at one time, but the segment number is not given.

SEGMENT NOT ON FILE

This occurs when a segment is referenced that has not been read in (Section I-C23).

SEGMENT NUMBER TOO HIGH

The segment number exceeds the total number of segments allowed (Section I-C23). The dimension parameter DSEG should be increased in a global edit. The suggested value is adequate only for the present segment number.

<SIZE> IS REQUIRED

Three (two in 2D) integers must be given to define the block size when the indirect addressing mode is not used (Section I-C2).

<SIZE> POINT NOT SET

A single integer has been given for SIZE as a point number in the indirect addressing mode (Section I-C2), but the indices for that point have not been set by a prior ITEM="POINT" statement (Section I-C23).

<SPAVAL> IS REQUIRED

This occurs when spacing is being set on a section for Hermite interpolation, or for ORTHOG points, but no value is given for the spacing.

<SPLINE> SURFACE CANNOT BE A ---

This occurs when the spline section defined by ISTART and IEND for a NEUMANN or EXTRAP section is a line in 3D or a point in 2D (Section I-C9, C26). The spline section must be a surface in 3D and a line in 2D, i.e., only one pair of corresponding entries in ISTART and IEND must be equal in 3D and only two pairs in 2D. One probable cause is the omission of ISTART and IEND when the NEUMANN or EXTRAP section is a line in 3D or a point in 2D.

SPLINE SURFACE SMALLER THAN NEUMANN SECTION

This occurs when the surface indicated by ISTART and IEND to be splined is smaller than the NEUMANN or EXTRAP section (Section I-C9, C26). ISTART and IEND can simply be omitted if the spline section is to coincide with the NEUMANN or EXTRAP section.

<START & END> ARE REQUIRED

This occurs when START and END have not been set either directly or indirectly (Section I-C23).

<START> IS OUT OF BLOCK

A value for START that is out of the block has been given or obtained from storage.

<START> POINT NOT SET

This occurs in the indirect addressing mode (Section I-C23) when a point number (a single positive integer) is given for START (instead of three indices) the indices corresponding to the numbered point have not been set by a prior ITEM="POINT" statement.

SUB-BLOCKS DO NOT COVER BLOCK

This occurs when the sub-blocks defined do not completely cover the block, as must be the case (Section I-C16). Define some additional sub-blocks.

SURFACE IS TOO LARGE

This occurs in the curved surface mode (Section I-C24) when the surface exceeds the dimensions provided. Increase DIMC globally.

SURFACE SIZE HAS NOT BEEN SET

This occurs in the curved surface mode (Section I-C24) when the size of the surface is not given by including two integers for SIZE.

TOO MANY ---

The indicated dimension has been exceeded (Section I-A5). The dimension parameter indicated should be changed in a global edit. The value indicated is the value needed in this particular instance, and it is possible that a later demand may be found that is greater. This message in regard to DMRB refers to the total number of points on all six (four in 2D) sides of a block.

TOO MANY POINTS IN BLOCK

This indicates that a point number in the indirect addressing mode (Section I-C23) exceeds the maximum number of numbered points allowed. Increase DPNT in a global edit. The suggested value is adequate only for this point number.

TOO MANY POINTS INPUT AT ONE TIME

This occurs when points on a segment are being input directly from the NAMELIST (Section I-C3) and the dimensions of the holding array are exceeded. Increase DIMP globally.

TOO MANY POINTS ON INPUT FILE

This occurs when all segments for a block are read from a file at one time (Section I-C23) and the total number of points on the file exceeds the maximum allowed in the holding array. The dimension parameter DIMD should be increased in a global edit. The suggested value is adequate only for this file.

TOO MANY POINTS ON FILE

This occurs when all segments for a block are read from a file at one time (Section I-C23) and the total number of points on the file exceeds the maximum allowed for a block. The dimension parameter DIMT should be increased in a global edit. The suggested value is adequate only for this block.

TOO MANY TERMS

This occurs when a value is being calculated to be stored (Section I-C1), and the number of terms in the calculation exceeds the maximum allowed. Increase the dimension parameter NVALMX in a global edit. The suggested value is adequate only for the present instance.

TOO MANY VALUES STORED

This occurs when the storage location given for a calculated value exceeds the number of storage locations allowed. Increase the dimension parameter DVAL globally. The suggested value is adequate only for the present instance.

<VALOUT> IS REQUIRED

This occurs when a calculated value is being stored (Section I-C1), but no storage location is given.

<VALUES> IS REQUIRED

This occurs when points are being read from the NAMELIST (Section I-C3), but no points are given.

ZERO ---

This indicates that a zero is occurring as a division. The probable reason is that all points in a calculation region have the same value. Plot the algebraic grid for the block involved to check for bad points.

PART II - CODE OPERATION

A. INTRODUCTION

There are a number of coding features that are used throughout the code, and these are discussed first rather than in each occurrence in the code.

1. Basic Notation

As has been noted in Section I-A3, the three curvilinear coordinates, ξ^1, ξ^2, ξ^3 , are usually denoted in the code by the integers C1, C2, C3, or by C(1), C(2), C(3). Another notation used is IC1, IC2, IC3, or IC(1), IC(2), IC(3) particularly in reference to an image point. The integer CI is usually used for the index i when it identifies a curvilinear coordinate ξ^i , and the integer RI is used for i in reference to a Cartesian component x_i . The block number is usually the integer B, or NBLK in the subroutines, is IB when an image block is referenced.

2. Adjustable Dimension Parameters

The fundamental dimension parameters (all integers) are the following:

NS - number of surrounding layers for output. (0, 1 or 2)

BASE - base number to which SSD file numbers are added. (Block B goes on file BASE+B)

DIMT - maximum number of points allowed in a block. (including the surrounding layers)

DIMB - maximum number of blocks allowed.

DMRB - maximum number of points allowed on an entire block boundary. (all six sides, four sides in 2D).

DIMS - maximum number of points allowed on a block side.
 DIML - maximum number of points allowed on a block edge.
 DIMI - maximum number of image points allowed in a block.
 DSUB - maximum number of sub-blocks allowed in a block.
 DNEU - maximum number of Neumann boundary sections allowed in a block.
 DORT - maximum number of orthogonal boundary sections allowed in a block.
 DREF - maximum number of reflective boundary sections allowed in a block.
 DMNT - maximum number of Neumann boundary points allowed in a block.
 DMOT - maximum number of orthogonal boundary points allowed in a block.
 DIMR - maximum total number of points allowed in the entire field (all blocks) when all blocks are kept in core.
 DIMN - maximum total number of Neumann boundary points allowed in the entire field (all blocks) when all blocks are kept in core.
 DIMO - maximum total number of orthogonal boundary points allowed in the entire field (all blocks) when all blocks are kept in core.
 DIMG - maximum number of image points allowed in the entire field (all blocks) when all blocks are kept in core.
 DIMP - maximum number of points that can be included on one ITEM="LIST" input statement.
 DIMD - maximum number of points that can be read from an entire file read at one time.
 DPNT - maximum number of numbered points allowed in a block.
 DSEG - maximum number of numbered segments allowed on an entire file read at one time.
 DIMC - maximum number of points allowed defining a curved surface on which a 2D grid is to be generated.
 NVALMX - maximum number of terms that can be included in calculated input parameters.
 DVAL - maximum number of calculated input parameters that can be stored.

All of these parameters occur in several routines and hence must be changed by global edits. The four parameters, DIMR, DIMN, DIMO, and DIMG are relevant only when all blocks are kept in core. When file storage is used for the individual blocks, all four of these parameters should be set to unity.

The code computes the size actually needed for each of the larger arrays and prints a comparison of the storage actually used and that allotted by the dimension statements.

3. Field Arrays

The nine basic single-block field arrays comprise COMMON/FIELD/. The arrays with values for each points in a block are the following:

R(3,DIMT) - Cartesian coordinates of grid points. x_i , $i=1,2,3$

P(3,DIMT) - Control functions. P_i , $i=1,2,3$

SPACE(3,DIMT) - Arc length spacing for control functions. s_i , $i=1,2,3$

RAD(3,DIMT) - Radius of curvature for control functions. ρ_i , $i=1,2,3$

IMAGE(0:3,DIMT) - Image point: block (first subscript 0) and curvilinear coordinates therein (ξ^i , $i=1,2,3$ for first subscript)

ACC(DIMT) - Optimum acceleration parameters for SOR iteration.

TYPE(DIMT) - Point classification.

In addition there are two arrays with values for each point on certain sections of the block boundary:

RR(3,0:3,DMNT) - Cartesian coordinates (second subscript 0), first derivatives on boundary (second subscript 1 and 2), and cross derivative on boundary (second subscript 3) for Neumann boundary points.

RO(3,0:3,DMOT) - Cartesian coordinates on a boundary section involved in iterative adjustment of control functions for orthogonal boundary points.

These nine field arrays are passed to most subroutines as arguments, in which case R, P, SPACE, and RAD appear in the subroutines in the following form:

R(3,SSDB,IS:DIM1,IS:DIM2,IS:DIM3)

where SSDB=1, IS=1-NS, and the integers DIM1,DIM2,DIM3 are received as arguments that have been given the values CMAX(i,block)+NS for i=1,2,3, respectively. Similarly, IMAGE appears as

IMAGE(0:3,SSDB,IS:DIM1,IS:DIM2,IS:DIM3)

and ACC and TYPE appear as

ACC(SSDB,IS:DIM1,IS:DIM2,IS:DIM3)

The arrays RR and RO appear as

RR(3,0:3,DMRR,SSDB)

RO(3,0:3,DMRO,SSDB)

where the integers DMRR and DMRO are received as arguments with the values DMRR(block) and DMRO(block) from integer arrays in the calling routine. (In all nine of these arrays, the dimension SSDB is a legacy of an earlier version without disk access and is now always equal to unity.)

Another related COMMON block, /SSDFL/, is also included to provide large one-dimensional storage arrays corresponding to each of the above nine single-block arrays. In addition, there is the array LINKF, corresponding to the single-block array LINK in COMMON/ILINK/, which serves to link the image points with corresponding object points. These stor-

age arrays are capable of holding all points in all blocks, and values for the block being treated can be written from the storage array into the single-block array and vice versa. These storage arrays have the same names as the single-block arrays but with an F appended to the end of the name, e.g. RF corresponds to R. The storage arrays RF, PF, SPACEF, and RADF are dimensioned DIM2F; IMAGEF is dimensioned DIM3F; ACCF and TYPEF are dimensioned DIMR; RRF and ROF are dimensioned DIM4F and DIM5F; and LINKF is dimensioned DIM6F. These dimensions (all integers) are calculated in PARAMETER statements from DIMR, DIMN, and DIMO. Also in this COMMON is the array NDEXF(6,DIMBF) which contains the location in these storage arrays for the first point in each block. The integer DIMBF is calculated in a PARAMETER statement from DIMB.

Several other COMMON blocks are used to transfer parameters between routines:

/PARAM/ - dimension parameters (all integers):

DMRR(DIMB) - number of Neumann boundary points in a block.

DMRO(DIMB) - number of orthogonal boundary points in a block.

CMAA(3,DIMB) - number of points in each of the three directions in a block.

BMAX - total number of blocks.

/INTERP/ - interpolation parameters (all integers, except CSCAL):

PROTYP - transfinite interpolation form.

PROJ(3) - interpolation directions.

PROF(3) - interpolation types.

BLEND(3) - blending function types.

CSCAL - radius of curvature scale. (real, not related to interpolation)

/SECTN/ - section parameters (all integers):

NAVG - block number of current block being treated.

B - block number. (NBLK in the subroutines)

START(3) - first corner of section.

END(3) - opposite corner of section.

S(3) - order of curvilinear coordinates.

RS(3) - order of Cartesian components.

/NEUMANN/ - Neumann boundary parameters (all integers):

IDXRR(DIMB,DNEU) - index array giving location in RR of first point in each section.

NEUNOR(DNEU,DIMB) - constant curvilinear coordinate on section.

NEUBON(3,0:1,DNEU,DIMB) - curvilinear coordinates of section corners (second subscript 0 for first corner, and 1 for opposite).

NEUNUM(DIMB) - total number of sections in a block.

/ORTHOGL/ - orthogonal boundary parameters (all integers):

IDXRO(DIMB,DORT) - index array giving location in RO of first point in each section.

ORTNOR(DORT,DIMB) - constant curvilinear coordinate on section.

ORTBON(3,0:1,DORT,DIMB) - curvilinear coordinates of section corners (second subscript 0 for first corner, and 1 for opposite)

ORTNUM(DIMB) - total number of sections in a block.

/REFLECT/ - reflective boundary parameters (all integers, except for REFDIR and WEIGHT).

REFNUM(DIMB) - total number of sections in a block.

REFNOR(DREF,DIMB) - constant curvilinear coordinate on section.

REFBON(3,0:1,DREF,DIMB) - curvilinear coordinates of section corners (second subscript 0 for first corner, and 1 for opposite).

WEIGHT - control for volume weighted average for special points (not related to reflective boundaries).

REFDIR(3,DREF,DIMB) - Cartesian components of normal to boundary.

/SSDCOM/ - block storage parameters (all integers):

SB - now a dummy, kept at unity.

LAST - number of blocks currently in core.

KSTORE - block storage indicator.

IMGF(DIMB) - indicator for image points in block interior.

/ILINK/ - image-object point linkage (all integers).

LINE(DIMB) - total number of image points in a block.

LINB(DIMB) - total number of blocks containing object points of image points in a block.

LINL(2,DIMB,DIMB) - first (first subscript 1) and last (first subscript 2) locations of a group of image points with all object points in a single block.

LINK(-3:3,DIMI) - curvilinear coordinates of image point (first subscript -1,-2,-3), object block number (first subscript 0), and curvilinear coordinates of object point (first subscript 1,2,3).

/SURFACE/ - curved surface on which 2D grid is generated:

RE(3,0:5,DIMC) - surface spline. As RR and RO, but with the second derivatives also (second subscript 4 and 5).

VR(3,0:2,0:2) - interpolated c_u and c_v for second and third subscripts 1,0 and 2,0; c_{uu} and c_{vv} for 1,1 and 2,2; c_{uv} for 1,2.

SMAX(2) - curved surface dimensions.

4. Index Functions

Several statement functions, with integer arguments, are used to determine the locations of points in field arrays or other quantities:

NX(B,C1,C2,C3) - location of point with curvilinear coordinates $\xi^i = C_i$, ($i=1,2,3$) in block B in single-blockarray.

NRB(B) - total number of points on all boundaries (all six sides) of block B.

NCMAX(B) - total number of points in block B.

NSPL(C1,C2), with $\xi^i = C_i$ ($i=1,2$) - location of a point on the curved surface on which a 2D grid is generated.

5. Block Storage

The code is set up to treat one block at a time, and hence the subroutines operate with only a single block in the nine field arrays R,P,SPACE,RAD,IMAGE,ACC,TYPE,RR,RO, and in LINK. Most of the subroutines are called for each block and hence use these arrays, with three individual subscripts for the three curvilinear coordinates. However, in the main program and in a few subroutines that involve loops over all blocks, (IMGIMG, SETIMG, FIXIMG, SETIMR, SETIMP, and SSD) these arrays have only a single subscript to identify the point location. In that case the location in the array is given by the index function NX(B,C1,C2,C3), where the arguments are the block number and the three curvilinear coordinate values for the point in the block.

The blocks are stored either on disk files, one block to a file, or in the core storage arrays in COMMON/SSDFL/. This is accomplished by subroutine SSD, and other subroutines beginning with SS for disk storage or CC for core storage, which are called whenever a new block is needed. The code keeps the number of the block presently in core as LAST and only accesses the storage when the next block to be treated is different from LAST. Subroutine SSD is called in each loop over the blocks to check and make this access if needed. (Since these calls are universal, they are not noted in the discussions of the various operations that follow.)

These subroutines either read and write the nine arrays in COMMON/FIELD/ and the array LINK to and from the disk files, or write these arrays to and from the storage arrays in COMMON/SSDFL/. With disk storage, the arrays are read and written with three subscripts for the point locations, while with core storage the single subscript form is used. In the latter case, the location of the start of each block in the storage arrays is in the array NDEXF(6,DIMBF), where the value with the first subscript equal to 1 is the location in the ACCF and TYPEF arrays, that with 2 is in the RF,PF,SPACEF, and RADF arrays, that with 3 is in the IMAGEF array, those with 4 and 5 are in the RRF and ROF arrays, respectively, and that with 6 is in the LINKF array. These locations are calculated simply as the appropriate multiples of the total number of points in the block (including the surrounding layers).

This construction was adopted in order to have an identical subroutine structure when blocks are kept on disk storage or all in core. No storage is wasted with this construction when disk storage is used, but storage for one extra block is used if all blocks are kept in core.

6. Kroneker Delta

The Kroneker delta,

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

is set in the integer array D(3,3):

$$D = 1,0,0,$$

$$0,1,0,$$

$$0,0,1$$

The Kroneker delta array serves to increment curvilinear coordinates in loops. Thus the arguments

$$C1 + D(1,N), C2 + D(2,N), C3 + D(3,N)$$

representing the triad (ξ^1, ξ^2, ξ^3) of curvilinear coordinates of a point, assume the following values for $N=1,2,3$:

$$N = 1: \quad C1 + 1, C2, \quad C3$$

$$N = 2: \quad C1, \quad C2 + 1, C3$$

$$N = 3: \quad C1, \quad C2, \quad C3 + 1$$

7. Cyclic Array

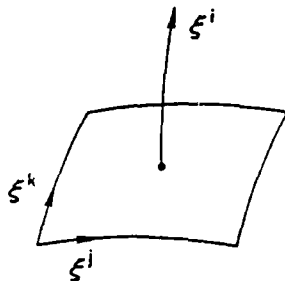
The integer cyclic array $CY(2,3)$ is

$$CY = 2,3,$$

$$3,1,$$

$$1,2$$

The cyclic array is used in circumstances where the two coordinate directions that are cyclic with that defined by a loop index are needed, e.g., in a cross product or on a surface.



Thus for $I = 1, 2, 3$, with $J = CY(1, I)$ and $K = CY(2, I)$, we have

$$I = 1 \quad J = 2, \quad K = 3$$

$$I = 2 \quad J = 3, \quad K = 1$$

$$I = 3 \quad J = 1, \quad K = 2$$

8. Loop Order

In many cases the order of three nested loops over the curvilinear coordinates is determined by the array $ORDER(3)$. The innermost loop is over the coordinate direction identified by the first entry, $ORDER(1)$, the next is over the coordinate given by $ORDER(2)$, etc. The limits of such loops are given in $START(3)$ and $END(3)$, where the entries $START(i)$ and $END(i)$ are the limits in the ξ^i direction.

9. Coordinate Derivatives

Throughout the code the Cartesian coordinate derivatives are usually placed in the array $DR(3, 0:3, 0:3)$ as

$$\left(\frac{r}{\xi^i} \right)_n = DR(n, i, 0)$$

$$(\underline{r}_{\xi^i \xi^j})_n = DR(n,i,j)$$

where n indicates the Cartesian component of the vector. (The indices i,j, and n all range from 1 to 3 here and below). These derivatives are evaluated by second-order central difference expressions in loops over the three directions and components using the Kroneker delta array D:

$$(\underline{r}_{\xi^i})_n = \frac{1}{2}[R(n,B,C1 + D(1,i), C2 + D(2,i), C3 + D(3,i))$$

$$- R(n,B,C1 - D(1,i), C2 - D(2,i), C3 - D(3,i))]$$

$$(\underline{r}_{\xi^i \xi^i})_n = R(n,B,C1 + D(1,i), C2 + D(2,i), C3 + D(3,i))$$

$$+ R(n,B,C1 - D(1,i), C2 - D(2,i), C3 - D(3,i))$$

$$- 2*R(n,B,C1,C2,C3)$$

$$(\underline{r}_{\xi^i \xi^j})_n = \frac{1}{4}[R(n,B,C1 + D(1,i) + D(1,j),$$

$$C2 + D(2,i) + D(2,j),$$

$$C3 + D(3,i) + D(3,j))$$

$$+ R(n,B, C1 - D(1,i) - D(1,j),$$

$$C2 - D(2,i) - D(2,j),$$

$$C3 - D(3,i) - D(3,j))]$$

$$- R(n,B, C1 + D(1,i) - D(1,j)$$

$$C2 + D(2,i) - D(2,j),$$

$$C3 + D(3,i) - D(3,j))$$

$$\begin{aligned}
& - R(n,B, C1 - D(1,i) + D(1,j), \\
& \quad C2 - D(2,i) + D(2,j), \\
& \quad C3 - D(3,i) + D(3,j)))]
\end{aligned}$$

10. Metric Elements

The elements of the covariant metric tensor are placed in the array $G(3,3)$:

$$g_{ij} = G(i,j)$$

and those of the contravariant (actually the product thereof with the square of the Jacobian) are in the array $GG(3,3)$:

$$gg^{ij} = GG(i,j)$$

These metric tensor elements are evaluated from the expressions

$$g_{ij} = r_{\xi i} \cdot r_{\xi j}$$

$$gg^{i\ell} = g_{jm}g_{kn} - g_{jn}g_{km} \quad \begin{matrix} (i,j,k) \\ (\ell,m,n) \end{matrix} \text{ cyclic}$$

B. INPUT PHASE

The code first sets the defaults for the variables in NAMELIST/INPUT/ and then accepts input statements to read that NAMELIST. The code loops through a series of such input statements, and the defaults are reset after each input statement. After reading an input statement, the code first retrieves any stored input values of SIZE, START, END, ISTART, and IEND that are indicated by negative entries (Section I-C1), and then checks the input variables for unreasonable values. The action taken on each input statement is described below, as determined by the value given to ITEM. This input phase is terminated by an input statement with ITEM="END".

The code sets values in START,END,ISTART, and IEND when numbered points are used (cf. Section I-C23 and II-B25), i.e., when only a single value is given for these quantities, before retrieving any stored values.

If ITEM="EXTRAP" (for extrapolated Neumann boundary conditions, Section I-26), SPAVAL is set to "EXTRAP" so that SPAN will be equal to "EXTRAP" on that section. This is used by NEUPTS to distinguish extrapolation boundary conditions from orthogonal boundary conditions.

1. ITEM="BLOCK" (Sets block size, Section I-C2)

The total number of blocks, BMAX, is incremented and the code stops if the dimension DIMB is exceeded. The code then checks the three entries in SIZE for compatibility with several dimensions, i.e., checks the limits on block size in regard to total number of points in the block, maximum number on a side and an edge, and the total number on the

entire block boundary. If all of the blocks are to be kept in core, the limit on the total number of points in all blocks is checked. Finally, the transfinite interpolation subroutine TRANS uses an EQUIVALENCE which imposes the limit

$$15 * DIMT \geq 12 * DIML ** 2$$

and this limit is checked. (This last limit could be obviated by removing the EQUIVALENCE between the arrays ARS and P in the main program, but at the cost of increased storage.)

The location of the block in the core storage arrays in COMMON/SSDFL/ is then calculated if all blocks are to be kept in core. Here the number of Neumann and orthogonal boundary points and the number of image points are temporarily set to unity since these numbers are not yet known.

The three entries in SIZE are then placed in the block size array, CMAX, for the current block number. Subroutine SSDW or CCDW is then called to initialize the block storage, and SSD is called to set LAST to the current block number. Then DEFAUL is called to default the field arrays.

The block size is set here only if SIZE is included (cf. Section I-C2). If a file number is included, all the boundary segments for the block are read from the file at one time as in the revision of Section II-B3.

2. ITEM = "SUB" (Sets up a sub-block, Section I-C16)

The counter SUBNUM is incremented for the block indicated by BLOCK, and the two opposing corners indicated by START and END, which define the sub-block, are placed in the arrays SUBSTA and SUBEND for that sub-block number.

3. ITEM="FILE" (Reads a section of points from a file, Section I-C3)

The file (given by the integer FILE) is first rewound if REWIND="YES", and READF is called to read the Cartesian coordinates of the points on the section identified by BLOCK, START, and END from the file.

If ALL="YES" is included, all segments on the file are read at once. This assumes the file was generated by the front-end code by ITEM="COMBINE" with CONTENT="YES", (cf. Section I-C23 and Vol. II, Section I-E21). In this case the segment table of contents is read from the file first, and then READB is called to read the entire file, placing the Cartesian coordinates of the points in the array BDATA(3,DIMD) where the second subscript counts the points. The location in this array of the first point on the segment is placed in the array LSEG(DSEG,0:2) with the second subscript 0, and the dimensions of the segment are placed with the second subscript 1 and 2. The segment number (the COREOUT number from the front-end code, Vol. II, Section I-E21) is the first subscript.

4. ITEM="LIST" (Reads a section of points from the NAMELIST, Section I-C3)

Subroutine READL is called to read the Cartesian coordinates of the points on the section identified by BLOCK, START, and END from the NAMELIST. The Cartesian coordinates appear on the input statement in the array VALUES.

5. ITEM="FIX" (Classifies a section of points as fixed boundary values, Section I-C4)

Subroutine SETYPE is called to set TYPE equal to ITEM on the section identified by BLOCK, START and END.

6. ITEM="OUT" (Classifies a section of points as out of the computation, Section I-C5)

Subroutine SETYPE is called to set TYPE equal to ITEM on the section identified by BLOCK, START, and END.

7. ITEM="NEUMANN" (Sets up a section of points for boundary orthogonality through Neumann boundary conditions, Section I-C9)

Subroutine SETNEU is called to set up the section identified by BLOCK, START, and END.

8. ITEM = "NEUMAN1" or "NEUMAN2" or "NEUMAN3" (Restricts movement of Neumann boundary points to one direction, Section I-C9.)

Subroutine SETRES is called to set up the restricted section identified by BLOCK, START, and END.

9. ITEM="ORTHOG" (Sets up a section of points for boundary orthogonality through iterative adjustment of control functions, Section I-C10.)

Subroutine SETORT is called to set up the section identified by BLOCK, START, and END.

10. ITEM="REFLECT" (Sets up a section of points for mirror-image boundary conditions, Section I-C8)

Subroutine SETREF is called to set up the section identified by BLOCK, START, and END.

11. ITEM="CUT" (Sets up a section of points as a branch cut, Section I-C7.)

If the cut is a line in 3D, or a point in 2D, IMONLY is set to 1 to indicate that no adjacent image points are to be set up. Subroutine SETIMO is called to set the image points in the object block, and SETIMI is called to set the image points in the image block for the cut specified by BLOCK, START, and END and IBLOCK, ISTART, and IEND. Subroutine MOVEIN is called to contract the section if appropriate.

The code checks for FIELD points that would be the objects of FIX, ORTHOG, or NEUMANN points, and reverses the imaging so that the FIELD point becomes an IMAGE point. The check is made in subroutine SETIMI, with the corresponding points to be changed being returned in LINK. The total number of points to be changed is NCHG.

12. ITEM="FIELD" (Classifies a single point as a point to be treated by the grid generation system, Section I-C12)

Subroutine SETYPE is called to set TYPE equal to ITEM for the point specified by BLOCK and POINT.

13. ITEM="IMAGE" (Classifies a single point as an image point and sets up the correspondence with an object point, Section I-C12.)

Subroutine SETOBJ is called to set up the correspondence between the image point specified by IBLOCK and IPOINT with the object point specified by BLOCK and POINT.

14. ITEM="AVERAGE" (Classifies a single point to be set as an average of all adjacent points, Section I-C12)

The value given for WEIGHT is saved in NWT, and WEIGHT is reset to "NO". Subroutine SETYPE is called to set TYPE equal to ITEM for the point specified by BLOCK and POINT.

15. ITEM="INTERP" (Sets Cartesian coordinate or control function values for points on a section by interpolation, Section I-C6.)

If FUN="POINTS", Cartesian coordinate values for all points on the section identified by BLOCK, START, and END that are not already classified as "FIX" are set by transfinite interpolation from the section boundaries by calling TRANS. If CLASS is equal to "FIX", then TYPE is reset to "FIX" after the interpolation by calling SETYPE. If FUN="CONTROL", it is the control functions that are determined by the interpolation. The blending functions are set to be linear regardless of the input when the interpolation is for the control functions.

If the required points are not specified on the entire boundary of the section, the interpolation is delayed until after the input is completed. In this case the interpolation parameters are stored on file 10 for later use. The total number of delayed sections will be NTER.

16. ITEM="SMOOTH" (Smooths Cartesian coordinates or control functions on a section, Section I-C13)

If FUN="BOUND", the Cartesian coordinates at all points within the section identified by BLOCK, START and END that have TYPE="FIELD" are smoothed immediately by calling SMOOTH. The smoothing is done in the curvilinear coordinate directions given in SMODIR, and only the Cartesian components given in SMOCOM are smoothed. If the section lies on a block boundary, smoothing is only done in the two directions on the boundary. Smoothing in the third direction in 2D is prevented.

If FUN="POINTS", the smoothing is deferred until after the interpolation for the initial algebraic grid. In this case the counter NSMOP is incremented and the relevant parameters are written on file 9, and no other action is taken during the input phase. Since image points will have been set by the time this smoothing is done, smoothing normal to the block boundaries is not prevented.

If FUN="CONTROL", the control functions within the sections will be smoothed after they have been calculated. Here also the relevant parameters are written on file 9, and no other action is taken during the input phase. Control functions are smoothed only in the two directions that are different from that of the function, e.g., P_1 is only smoothed in the ξ^2 and ξ^3 directions.

In the last two cases, if ALL="YES" the entire field, i.e., all points in all blocks that have TYPE="FIELD", will be smoothed. Here for each block the appropriate counter is incremented and the relevant parameters are written on file 9.

17. ITEM="CONTROL" (Calculates control functions on a section, Section I-C15)

The control functions are calculated at all points on a section defined by BLOCK, START, and END. This is done by calling CONLINR if the section is a line, or by calling CONSURR for a surface, if CONTYP is equal to "RADIUS". Otherwise CONLINE or CONSURF is called. (Since the control functions are calculated for all blocks after the initial algebraic grid has been generated, this present feature is rarely used, and then only in very unusual cases.)

18. ITEM="INITIAL" (Sets the type of interpolation to be used for the initial algebraic grid, Section I-C14)

If ALL="YES" the same type of interpolation is set for all blocks. Otherwise the type is set only for the block specified by BLOCK. The interpolation parameters, PROFOR, PRODIR and BLEND, are stored in IPROF, IPROJ, and IBLEND for later use after the input phase.

19. ITEM="SPACE" (Sets the off-boundary spacing for a section for boundary orthogonality, Section I-C11)

Subroutine SETSPA is called to set the spacing on the section identified by BLOCK, START, and END.

20. ITEM="SETVAL" (Calculates and stores certain input quantities, Section I-C1.)

A sum, product, a sum-minus-one, difference, and difference-plus-one, of terms input in TERMS is calculated and stored in the array IVALUE at the location indicated by VALOUT for later use as a value in SIZE, START, END, ISTART, or IEND. The input parameter MATH determines the type of calculation. Entries in TERMS indicated by negative numbers are themselves retrieved from storage.

21. ITEM="STORE" (Sets the storage file for the final grid, Section I-C17)

If file is equal to "RESTART", the indicator IRESTRT is set to "YES" to cause a restart file to be written on file 7. Otherwise this input operation sets the storage file number, given by FILE, in GRIDFIL for storage of the entire grid for later use in a PDE code. (This is not to be confused with file 8 on which sections of the grid may be written for plotting.)

22. ITEM = "SIZE" (Sets block size, Section I-C2)

The block size is set as in Section II-B1, but the block counter is not incremented.

23. ITEM = "UNFIX" (Returns classification of a section of points to the default, "FIELD", Section I-C4)

This operates as does operation ITEM="FIX" (Section II-B5), but the setting is to "UNFIX", instead of "FIX".

24. ITEM = "EXTRAP" (Sets up a section of points for zero curvature boundary conditions, Section I-C9)

This operates as does ITEM="NEUMANN" (Section II-B7), but the setting is to "EXTRAP", instead of "NEUMANN".

25. ITEM="POINT" (Sets a numbered point, Section I-C23)

The three indices (ξ^1, ξ^2, ξ^3) for a numbered point are set either directly or relationally from another point. If ALL="YES" a group of points is set from a group of other points. The three indices are stored in the array LPOINT(3,DPNT,DIMB), where the point number and block number appear as the second and third subscripts. The indices are set directly to the values in LOCAT(3) if it is included. Otherwise, the indices are incremented from those of the point indicated by OPOINT. If SEGMENT is present, the increment is placed in POINTS from the segment array LSEG (cf. revision of Section II-B3).

26. ITEM="SEGMENT" (Reads a section of points, Section I-C3)

The subroutine READS is called to read the Cartesian coordinates of the points for the numbered segment indicated by SEGMENT from the storage array BDATA onto the section identified by BLOCK,START, and END.

27. ITEM="SURFACE" (Reading of curved surface from file, Section I-C24)

The Cartesian coordinates of the surface are read from the indicated file, or from the namelist, into the array RE(3,0,NSPL(C1,C2)). MODE is set to "SURFACE".

C. SETUP PHASE

1. Derivative Forms

The integer indicators AFIRST and ACROSS are set according to the form of the derivatives specified by DFIRST and DCROSS on the input (Section I-C20).

2. Sub-Blocks

In a loop over all the blocks, if no sub-blocks have been specified for a block, the sub-block corners in SUBSTA and SUBEND are set to the entire block. Otherwise CHKSUB is called to check the sub-block structure for the block for complete coverage of the block (Section I-C16).

3. Restart

If the indicator RESTART is different from the default value "NO", (Section I-C22) certain quantities are read from file 7, and IPROJ is set to "NONE" to skip the generation of the algebraic grid. A loop is made over all blocks, calling REDRES for each block to read the field arrays for the block and then calling SSDW or CCDW to set up the storage. If the restart is from the midst of the iterative solution, the code then skips to continue the iteration.

4. Image Points

The code calls IMGIMG to remove ambiguities arising from image points having other image points as objects to set the image points for special points.

5. Boundary Array for Neumann Boundaries

The Neumann boundary sections for a block, B, are stacked in the array RR(3,0:3,n), with the last subscript counting the points from 1 to the total number DMRR(B). The location in RR of the first point in each section, i.e., the value of n, is in the array IDXRR(B,m), where m is the section number. Points are located in RR by the index function

$$NDXRR(B,m,i,j) = IDXRR(B,m) + (j-1) * IT + i$$

where i and j are the increments from the first corner in the two coordinates that vary on the section, in cyclic order with the one that is constant thereon, and IT is total number of points in the i-direction on the section.

The total number of Neumann sections in the block is in the array NEUNUM(B), and the total number of points in all these sections is in the array DMRR(B). Each of these sections is defined on input (Section I-C9). The second subscript in RR identifies (1) the Cartesian coordinates of the point on the section; (1 and 2) the first derivatives in the two directions on the section, cyclic with the curvilinear coordinate that is constant on the section; and (3) the cross-derivative on the surface. The first subscript of RR identifies the Cartesian component. In addition, there are two other arrays, NEUNOR(m,B) which identifies the curvilinear coordinate that is constant on section m and NEUBON(3,0:1,m,B) which contains the curvilinear coordinates of the two opposite corners (0 and 1 for the second subscript) defining the section (Section II-B7).

All blocks are swept and, for each having Neumann boundary sections, subroutine CHKNEU is called to check for bad points and NEUIDX is called to set up the boundary array for the block, and SETSPL is called to spline all the sections in the block. The next available location in the core storage array RRF is placed in NDEXF with the first subscript 4 (Section II-A5).

6. Boundary Array for Orthogonal Boundaries

The orthogonal boundary sections for a block are stacked in the array RO (3, 0:3,n) in the same manner described above for Neumann sections in Section II-C5, except that only the Cartesian coordinates are included here. All the other arrays in that section occur here also, with RR replaced by RO, and with NEU by ORT, in all array names and in the name of the index function. The input of the sections is described in Section I-C10 and the resulting action in Section II-B9.

All blocks are swept and, for each having orthogonal boundary sections (based on iterative adjustment of the control functions), subroutine CHKORT is called to check for bad points and ORTIDX is called to set up the orthogonal boundary array for the block. The next available location in the core storage array ROF is placed in NDEXF with the first subscript 5 (Section II-A5).

7. Interpolation for Initial Algebraic Grid

If an initial grid is not being read in, an initial algebraic grid is generated by transfinite interpolation in each block by calling TRANS with PROTOP="FACES" and with the interpolation directions and blending

functions that were specified by PRODIR and BLEND on the ITEM="INITIAL" read statement during the input phase (Section I-C14). This interpolation sets values for the Cartesian coordinates at all points in each block that are not classified "FIX".

Subroutine IMGPTS is first called to set the values of the Cartesian coordinates at the image points in all the blocks. If some section interpolation has been delayed from the input phase (NTER=0, cf. revision of Section II-B15), then file 10 is rewound. The interpolation parameters for each of the delayed sections are read from file 10 in succession and, for each, TRANS is called to do the interpolation. Subroutine SETYPE is called to classify the points on the section as "FIX" after the interpolation if such is indicated.

In the delayed section interpolation, and in the interpolation for the algebraic grid that follows, if the required values for the Cartesian coordinates have not been set on the entire boundary of the interpolation region, interpolation of lesser dimensionality is attempted to supply the missing values.

If TRANS finds insufficient boundary values in a certain interpolation direction, that direction is returned in LUS(1). Whether the missing values are on the lower or upper boundary of the section in that direction is indicated by a 0 or 1 value in LUS(2). After saving the values of START,END, and PRODIR(1), START and END are reset to the section boundary with the missing data and PRODIR(1) is set to 0 to cause the interpolation to default to the dimensionality of that section boundary. Subroutine TRANS is then called again to interpolate for values on that section boundary.

If the section boundary with the missing values is a surface, and TRANS again detects missing data, the process is repeated with still another call to TRANS to interpolate on an edge of that surface. In that case, after all such edges have been interpolated, the code returns to the surface and again calls TRANS for that surface.

This process continues until all missing data has been set or until an unresolvable situation is encountered, in which case LUS(2) is set to -1.

If a sub-block structure (Section I-C16) has been set up for a block, CHKINT is called to check for interior points at which values have been set. If such points are present, the interpolation is done by sub-blocks, i.e., within sub-blocks defined by the opposing corners set in SUBSTA and SUBEND. Subroutine IMGPTS is called before the interpolation to set values of the Cartesian coordinates at "IMAGE" points.

For 2D, the x_3 coordinate on the adjacent layers at 0 and 2 in the third direction are set to ± 1 in such a way as to make the system right-handed (with $x_3=0$ on $\xi^3=1$) by calling SETAXS. The x_1 and x_2 Cartesian coordinates at points on the adjacent layers are set equal to those values at 1 after the interpolation by calling SETH2D.

8. Normals on Reflective Boundaries

The reflective boundary sections for a block use a set of arrays similar to those for Neumann sections in Section II-C5, but with NEU replaced by REF in the names. There is an additional array, REFDIR(3,m,B), which contains the Cartesian components of the unit

normal to the section, and there is no counterpart of the array RR. The input of these sections is described in Section I-C8 and the resulting action thereon in Section II-B10.

All blocks are swept, and for each having reflective boundary sections, subroutine SETNOR is called to set the normals to the sections.

9. Values At Image, Reflect, Neumann, and Average Points

The values of the Cartesian coordinates at the image reflect, Neumann, and average points are then set on the initial grid by calling SETIMR(1). If there are any special points to be set by a volume-weighted average, SETIMR(1) is called a second time in order to use the relevant cell volumes from the first call.

10. Smooth Coordinates

If smoothing of the initial grid has been called for during the input phase (with ITEM="SMOOTH", Section I-C13) then the smoothing parameters are read from file 9 after rewinding. The smoothing is done by calling SMOOTH, using P as the dummy array. Since only the "FIELD" points are smoothed, SETIMR(1) is again called to reset the values at the image points.

11. Jacobian Check

All blocks are swept, calling subroutine CHKSET to check for "FIELD" or "IMAGE" points adjacent to points for which no Cartesian coordinate values have been set. Next JACBCK is called to check the initial algebraic grids for points with zero Jacobian or a locally

left-handed system. The former usually means some input data has been omitted and hence the code stops when a zero Jacobian is found in the field (but not on a fixed boundary). Although a left-handed system could mean twisted boundaries as a result of errors in the input data, it could also mean simply that the algebraic grid has overlapped a segment of the physical boundary. In that case the elliptic generation system may be able to resolve the overlap and produce a grid that is right-handed everywhere. Therefore, the code notes the presence of the locally left-handed system but continues on unless a stop has been called for on the input (Section I-C19).

12. Coordinate Extremes

All blocks are swept, calling subroutine EXTCOR to evaluate and print the extreme values of the Cartesian coordinates of the points in each block.

13. Radius of Curvature Scale

A coordinate scale, RSCAL, is calculated in EXTCOR as the maximum extent of the entire physical region in any Cartesian direction. A point scale PSCAL is also calculated, as the cube root (square root in 2D) of the total number of points in all blocks. From these a scale, CSCAL, for the radius of curvature to be used in the evaluation of the control function is calculated as

$$CSCAL = \frac{RSCAL}{PSCAL} 10^4$$

This form arises from consideration of the curvature contribution r_ξ/r which appears in the control function (Appendix C). If an error ϵ in the control function is tolerable, the curvature contribution is negligible if the radius of curvature is larger than CSCAL, defined by

$$\frac{r_\xi}{\text{CSCAL}} < \epsilon$$

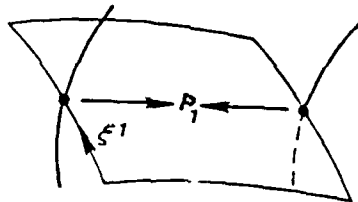
Also, from the arc length contribution to the control function, $r_{\xi\xi}/r_\xi$, it follows that when the spacing doubles from one point to the next, the control function is about $(2\Delta r - \Delta r)/\Delta r = 1$, and thus a nominal value for the control function in unity. Therefore, since RSCAL/PSCAL is a measure of the spacing, r_ξ , the scale for the radius of curvature is set as given above, corresponding to an error of about 0.01% in the control function. Surfaces with radius of curvature larger than CSCAL (e.g. flat surfaces with infinite radius of curvature) have their radius set to CSCAL by the code.

14. Control Functions

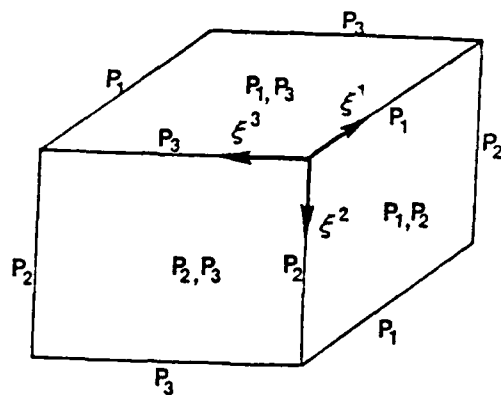
The three control functions, $P_i (i=1,2,3)$, are in the field array P. If CONTYP (Section I-C15) is equal to "ORTHO" or "ORTHO2", CONINT is called for each block to produce control functions from the initial grid, but with terms arising from nonorthogonality omitted. Otherwise the control functions are interpolated from boundary values by transfinite interpolation. (If the 2D curved surface mode is indicated, SCONINT is called, instead of CONINT, for control functions evaluated

from the algebraic grid.) If a sub-block structure has been set up for a block, this interpolation is by sub-blocks (Section I-C16). In any case the control functions are used only at "FIELD" points.

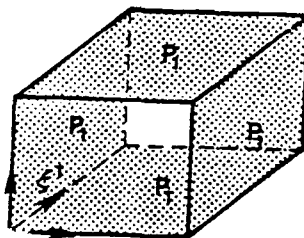
If CONTYP="SIDES", the control functions are first evaluated on all surfaces of a block by calling CONSURF for each side of the block. In this surface evaluation, only the two control functions in the curvilinear coordinate directions that vary on the surface are determined. Both the arc length and the curvature contributions to the control functions are included in this evaluation on the surface, the latter being interpolated from the edges onto the surfaces by calling TRANS.



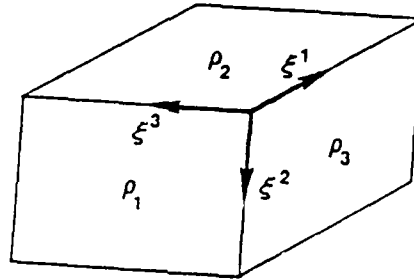
On the edges of the surface, only the single control function in the direction that varies along the edge is determined. The result is indicated in the following diagram:



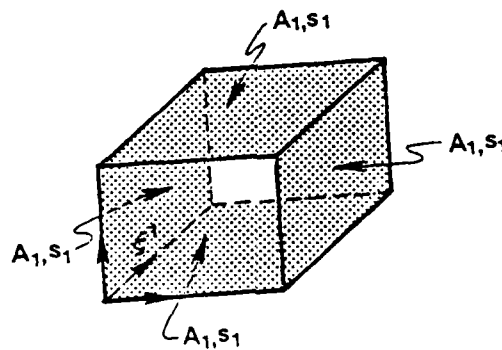
Each control function is then interpolated into the interior of the block from the four sides on which it is known by calling TRANS.



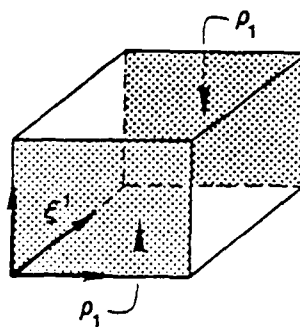
For CONTYP="RADIUS", the arc length contribution (placed in P) and the arc length spacing (placed in the field array SPACE) in the two curvilinear coordinate directions that vary on a surface are evaluated on each side of the block. This again is represented by the preceding diagram with each of these two quantities indicated by P_i there. The radius of curvature of the surface is evaluated on each surface also, as in the following diagram, and is placed in the field array RAD.



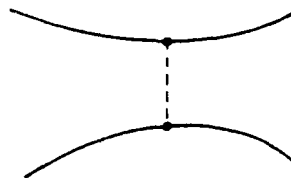
The evaluation of all three of these quantities is done by calling CONSURR for each side of the block. The arc length spacing, s_i , and the arc length contribution, A_i , to the control function are interpolated by TRANS into the interior of the block from the four sides on which they are known, using linear blending functions:



The radius of curvature, ρ_i , is interpolated by TRANS into the interior from the two sides on which it is known, using blending functions based on the hyperbolic sine (Appendix G).



If the radius of curvature is of opposite signs at the two corresponding points on a pair of opposing sides of the block,



an infinite radius of curvature must occur between the boundaries, and account of this is taken in the interpolation (Appendix G).

The control function is finally formed by adding the arc length spacing divided by the radius of curvature to the arc length contribution, this being done by calling CONFUN. Subroutine SETIMP is then called to set the control functions at the points not designated "FIELD" or "OUT", by extrapolation from the "FIELD" points.

After the control functions have been evaluated in all blocks, these functions are smoothed if such has been called for during the input phase. Subroutine CHKCON is called first for each block to check for unset values that are needed by the smoothing. The smoothing parameters are read from file 9 and SMOOTH is called, using R as the dummy array.

15. Control Function Extremes

All blocks are swept, determining and printing the extreme values of the three control functions over the entire composite grid by calling EXTCON.

16. Acceleration Parameters

If ACCPAR="OPTIMUM", (Section I-C18) then OPTACC is called for each block to calculate acceleration parameters for each "FIELD" point, these parameters being placed in the field array ACC. (If the 2D curved surface mode is indicated, SOPTACC is called, instead of OPTACC, to generate the optimum acceleration parameters.) Otherwise ACC is set to ACCPAR for all points by calling SETACC.

17. Orthogonal Boundary Initialization

If CONUPI is not 0 (Section I-C10), subroutine CONSETU is called for each block to initialize the orthogonal boundary sections.

18. Default of Control Functions

If CONTYP="INITIAL", smoothing of the control functions is set up.

19. Curved Surface Spline

If a 2D grid is to be generated on a curved surface, the Cartesian coordinates of the points on the surface are placed in the array SUR(3,0:5,DIMC) with a 0 for the second subscript. The last subscript counts the points according to the index function NSPL. Subroutine SPLSUR is then called to spline the surface, and the spline coefficients are placed in the surface spline array RE(3,0:5,DIMC) with values of the second subscript as follows:

0	r
1	r_u
2	r_v
3	r_{uv}
4	r_{uu}
5	r_{vv}

Account is taken of the direction of larger dimension on the surface in the interest of vectorization.

D. OUTPUT PHASE

There follows next a series of reads of the NAMELIST/OUTPUT/ through which the type of output to be given is specified. Each of these input statements contains ITEM="___", where the alphanumeric value given determines the type of output (Section I-D). The print of the initial grid is done in immediate response to the input statement. Data from an input statement calling for a print, plot, or storage of the final grid is saved for use after the grid has been generated. The quantities in this NAMELIST are defaulted after each input statement. This phase is terminated by an input statement with ITEM="END". The various output actions indicated are as follows:

As in the NAMELIST/INPUT/, numbered points can be used as single entries in START and END, (cf. Section I-C23).

On all the OUTPUT operations, the inclusion of PART="SIDES" causes all sides of a block to be output, and "EDGES" produces all edges.

1. ITEM="INITIAL" (prints initial grid, Section I-D1)

This prints all, or a portion of, the initial grid by calling PRTGRD. The entire grid is printed if ALL="YES". Otherwise the section identified by BLOCK, START, and END is printed.

2. ITEM="PRINT" (prints final grid, Section I-D1)

This prints the final grid in the same manner, and under the same type of control, as described for the initial grid. The only difference is that "OUT" points are not included in the default of this print. Again any number of input statements can be given, one for each section to be printed. The print is done by calling PRTGRD.

3. ITEM="PLOT" (writes final grid on file for plotting, Section I-D2)

This writes the final grid on file 8 in the same manner, and under the same type of control, as described above for printing the final grid. Again any number of input statements can be given, one for each section to be plotted. The write is done by calling WRTPLT.

4. ITEM="ERROR" (prints iteration error norms, Section I-D3)

This causes the maximum norm of the change in the Cartesian coordinates between iterations to be printed after each iteration. If BLKERR="YES", the norm in each block is printed, otherwise only a single norm over the entire field is printed.

E. GENERATION PHASE

1. Iteration

If ACCPAR=0 the initial algebraic grid is output as the final grid. Otherwise, the elliptic grid generation equations are solved by point Gauss-Seidel iteration. The SOR sweeps alternate in all three directions so that the iteration is symmetric point SOR. The control functions may be held fixed or may be changed iteratively to produce boundary orthogonality and specified off-boundary spacing (Section I-C10). If the control functions are to be changed iteratively to produce boundary orthogonality and specified off-boundary spacing, then after a specified number, CONUPI, of iterations, CONSETE is called for each block to change the control functions, and OPTACC is called to recalculate the control functions if such has been indicated.

Each iteration for the solution is done by a call to VOLSYS, after which SETIMR(0) is called to reset the values of the Cartesian coordinates at all "IMAGE", "REFLECT", "NEUMANN", and "AVERAGE" points, using the current values at the relevant "FIELD" points. In the 2D curved surface mode, SURSYS is called, instead of VOLSYS, to sweep the field.

The iteration error norm (the maximum change in any Cartesian coordinate on the field relative to the coordinate scale, RSCAL, Section II-B13) is then printed and checked for convergence. The code stops if this norm exceeds 1.0.

2. Output

When convergence is achieved, or if the maximum number of iterations allowed is reached, values of the Cartesian coordinates on the second surrounding layer of points around each block are set, if such has been called for, by calling SETIMR(NS). All or section of the final grid are printed and/or written on file 8 for plotting according to the instructions that have been given through the reads of NAMELIST/OUTPUT/ during the Output Phase (Section II-D) by calling PRTGRD or WRTPLT for each section indicated. If GRIDFIL is not zero, the grid is written on file for latter input to a PDE code. If OUTER is equal to "YES", the arrays R, TYPE, and IMAGE, including the surrounding layers, are all written on this file by calling WRTGRD, otherwise only R, without the surrounding layers, is written by calling WTRRRR. If a restart file has been called for, certain quantities are written on file 7 after re-winding, and WRTRES is called for each block to write the field arrays on file 7.

In the 2D curved surface mode, SURCOR is called for each block to convert the surface parametric coordinates to Cartesian coordinates. The curved surface itself is stored as the first section on file 8 for plotting. Sections of the grid stored by ITEM="PLOT" statements will follow the curved surface on that file.

F. SUBROUTINES

1. Classes

There are two basic classes of subroutines in the code. One class, composed of IMGIMG, SETIMR, SETIMP, IMGPTS, FIXIMG, BLKCON, and SSD, has the COMMON blocks FIELD and SSDFL as does the main program. In these six subroutines, and in the main program, the field arrays R,P,SPACE, RAD, IMAGE,ACC,TYPE, and LINK are dimensioned (_,DIMT), where the first dimension is 3 for the first four of these, 0:3 for IMAGE, non-existent for ACC and TYPE, and -3:3 for LINK. Similarly, the boundary arrays RR and RO are dimensioned (3,0:3,DMNT) and (3,0:3,DMOT). Points in the block are thus located in these ten arrays in a one-dimensional manner and must be accessed in all but LINK through the index function NX(B,C1,C2,C3) where B is the block number, and the last three subscripts are the curvilinear coordinates, ξ^i , of the point in the block. This function is defined as

$$\begin{aligned} \text{NDX} = & ((C1-IS) + ((C2-IS) \\ & + ((C2-IS) + (C3-IS) * (\text{CMAX}(2,B) + 2 * \text{NS}) \\ & * (\text{CMAX}(1,3) + 2 * \text{NS})) + 1 \end{aligned}$$

Here NS is the number of surrounding layers, and $IS=1-NS$, so that C1 runs from 0 to $\text{CMAX}(1,B)+NS$, etc. The first three of these subroutines and the main program contain loops over all blocks, calling SSD to place a block in core. Subroutines IMGPTS, FIXIMG, and SSD receive the block number as the argument. Only these routines and the main program can contain such loops over all blocks, and only these can call SSD or other related routines beginning in SS or CC. In the discussion of these

routines a loop over all blocks will always contain a call to SSD to put the block in core, and this will not be explicitly mentioned in the discussion.

All the other subroutines treat the seven field arrays R,P,SPACE, RAD,IMAGE,ACC, and TYPE as three-dimensional in the sense of point locations, so that points can be accessed directly by the three subscripts C1,C2,C3. In these subroutines these arrays are variable-dimension arrays received as arguments, and are dimensioned

$$(_,1,IS:DIM1,IS:DIM2,IS:DIM3)$$

where the first subscript is the same as used in the other class. Here IS is as defined for the other class, and DIM1,DIM2,DIM3 are received as arguments. Similarly the RR and RO arrays are also variable-dimension arrays received as arguments, and are dimensioned (3,0:3,DMRR,1) and (3,0:3,DMRO,1), with DMRR and DMRO received as arguments. These subroutines can only access the block that is currently in core, and the block number of this block is received as NBLK in COMMON/SECTN/.

2. SUBROUTINE SSD (access to storage)

This subroutine places a block in core by first either SSDW or CCDW to write the block currently in core out to storage, and then calling SSSDR or CCDDR to read the desired block into core. The block to be placed in core is identified by the argument B, and the block currently in core by LAST. If these are the same block, the routine simply

returns. If KSTORE = "FILE", reading and writing are done by SSDR and SSDW using the disk; otherwise CCDR and CCDW are used to access master storage arrays in core (Section II-A5).

3. SUBROUTINE IMGIMG (Resolution of image-object ambiguities)

This subroutine removes ambiguities arising from image points having other image points as object points, and also sets the image points associated with special points.

If the grid is comprised of only a single block, SSDW or CCDW is first called to write the block on file or in the core storage array. This is necessary since the TYPE array is read from the storage during the removal of ambiguities as discussed below.

Since it will often be the case that image points only occur on the block sides and surrounding layers, the interior of a block is skipped in the assignment of values at image points unless there is an image point in the interior. Therefore a loop over all blocks is made first, searching for image points in the interior. The presence of such interior image points is indicated by setting IMGF to 1 for the block in question. This value is used by subroutine LIM to set up the loops over image points used in this routine.

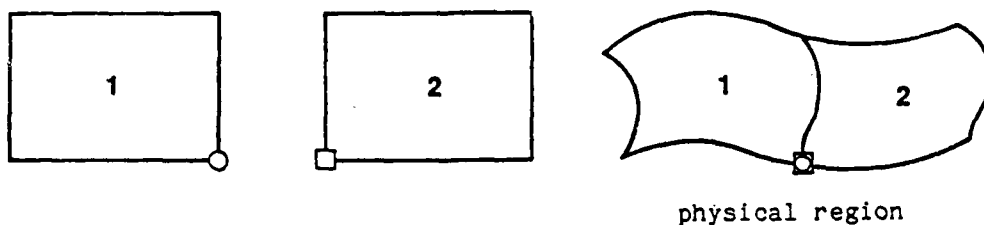
Special Points

Blocks to be searched for special points are placed in core by a call to SSD with the argument B (in a loop over all blocks), while blocks containing image points of the special points are placed in core by a call with argument IB.

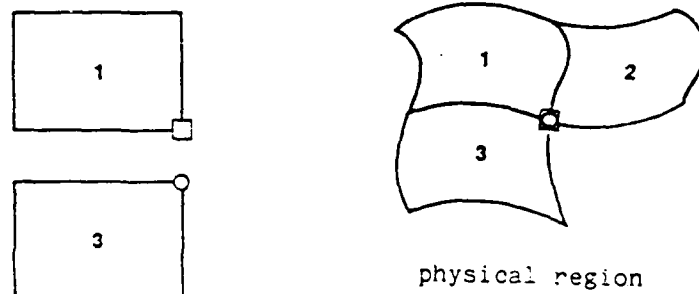
In each block a set of four nested loops over all points searches for points with TYPE equal to "FIELD" or "AVERAGE" for which a non-zero entry appears for the block number in the array IMAGE. Since only points having TYPE="IMAGE", i.e., image points, should have such a non-zero entry in IMAGE (IMAGE is defaulted to zero), it follows that the point in question must have been an image point which had its designation changed to make it a special point. Since the Cartesian coordinates at such special points are to be calculated by the grid generation system, the object point of this erstwhile image point is changed to an image point with the now special point as its object.

Ambiguities

It is possible during the input phase to have the object point of an image point become the image of some other object point. For instance if the right side of block 1 in the figure below is declared an object section of the left side of block 2 (object points are circles and images are squares on the diagram):



and later in the input the top side of block 3 is made the object of the bottom side of block 1:



then the lower-right corner point of block 1, which is the object point for the lower-left corner image point of block 2, has become an image point. Since the grid generation system is to operate only once at coincident points, the upper-right corner point of block 3 must be made the object point of both of these image points.

Therefore, a set of loops sweeps all points in, and on the surrounding layers of, all blocks, searching for points with TYPE="IMAGE" (i.e., image points) for which the object point also has TYPE="IMAGE". Since this means that the object point of the point in question is itself the image of some other object point, the object of the point in question is reset to this latter object point. The end result of this is that, although a single point may be the object of several image points, no image point will have another image as its object and no coincident points will appear as "FIELD" points in more than one place in computational space.

In this removal of ambiguities, a block to be searched is placed in core by a call to SSD with the argument IB in a loop over all blocks. Subroutine SETIMG is then called to first sort the blocks containing

object points of image points in the present block. All the object points in one block are considered before proceeding to the next. For each IMAGE point found, SSD3 or CCD2 is called with argument B to read the TYPE array for the block containing the object point into the array NTYPE if this block (B) is different from that (LASTB) containing the object point of the previous image point treated. If the object point is an image point, i.e., if NTYPE="IMAGE" for the object point, then the block number B is saved as IIB, and SSD is called with argument IIB to place the block containing this object point in core. Then B is reset to the block number of the block containing the object of this object point, and SSD3 or CCD2 is called with B to read the TYPE array for this last block into NTYPE. If NTYPE for this second object point is equal to "IMAGE" then still another object point is obtained in the same manner. When an object point is reached for which NTYPE is not equal to "IMAGE", SSD is called again with argument IB to place the original block being searched back into core (the signal for such being given by NEWTYP = 1). Finally, the values in the IMAGE array for the original image point are changed to make this last object point the object point of that image point.

Before returning, BLKCON is called for each block to check for possible loss of derivative continuity between blocks if such has been called for.

Point Classification Check and Sort

The routine calls CHKTYP for each block to check for "FIELD" points that are adjacent to "OUT" points, stopping if any are found. Subroutine FIXIMG is called to reclassify image points connected to "OUT" points as "OUT" points, and then to reclassify image points connected to anything other than "FIELD" points as "FIX" points. Subroutine SETIMG is then called to sort the image points for each block into groups having object points in a single block.

4. SUBROUTINE FIXIMG (Reclassification of image points)

This subroutine checks a block for image points that have other image points as object points, reclassifies image points having "OUT" points as object points as "OUT" also. This routine uses the arrays in COMMON/FIELD/ and hence refers to point locations through the index array NX as does the main program.

The routine functions as does subroutine IMGPTS, but reads the array TYPE, instead of R, from storage into the array NTYPE, instead of RAD, by calling either SSD3 or CCD2. Also the changes discussed above are made in TYPE for the block indicated by the argument IB, instead of setting values in R. Points that have "IMAGE" or "OUT" points as objects are checked for adjacent "FIELD" points, and the code stops if any are found.

5. SUBROUTINE IMGPTS (Cartesian coordinate values at image points)

This subroutine sets values of the Cartesian coordinates at image points equal to the values at corresponding object points. This routine uses the arrays in COMMON/FIELD/ and hence refers to point locations through the index arrays NX as does the main program.

For a single block case (BMAX=1), the routine first writes the block to storage by calling either SSDW or CCDW. This is necessary because the Cartesian coordinate values for the object points must be read from storage as explained below.

The key arrays involved in the image-object point correspondence here are LINK, LINL, and LINB, which are set by subroutine SETIMG, and are explained for that subroutine. The image points for the block have been sorted so that all those having object points in a single block are together in order to minimize the necessary accesses to storage.

The routine makes a loop over all blocks containing object points associated with image points in the block indicated by the argument IB. The object block number is obtained from the array LINK, with the first subscript 0, for the first image point in each group (from LINL), and the Cartesian coordinate array, R, for this block is read into the array RAD by calling either SSD1 or CCD1. The curvilinear coordinates of the image point (IC1,IC2,IC3), and those of the corresponding object point (C(1),C(2),C(3)), are obtained from the array LINK, and the values for the Cartesian coordinates in RAD for the object point are placed in R for the image point.

6. SUBROUTINE SETIMR (Cartesian coordinate values at boundary points)

This routine sets the Cartesian coordinate values at image points equal to the current values at the corresponding object points if the argument MS is not 0, and sets values at Neumann and reflective boundaries, special points, and on the adjacent layers for the 2D mode in any case. If BMAX is 1, either SSDW or CCDW is first called to write the single block on storage. This is necessary because values from the object block are read from storage as discussed below.

Average Points

In the setting of values at the special points designated as AVERAGE points, a loop is made over all blocks, each being placed in core by a call to SSD with argument B. The block is searched for AVERAGE points, and the surrounding FIELD points in the block are found and recorded in the arrays P and RAD for each AVERAGE point. Then a second loop is made over all blocks, each being placed in core by a call to SSD with argument IB. The block here is searched for IMAGE points having AVERAGE points as objects. For each such IMAGE point, the block containing the object point is placed in core by calling SSD with argument B. If the object point is an AVERAGE point, SSD is called with argument IB to place the block containing the image point back in core. The FIELD points surrounding this image point are then located, and SSD is called with argument B to place the block containing the AVERAGE point back in core so that these surrounding points can be added to those already found for this AVERAGE point.

Here SSD1 or CCD1 is called to read the coordinate array R into the array SPACE for block IB. The surrounding points are then recorded in the arrays P and RAD for block B. A final loop over all blocks is made to set the values at the AVERAGE points as the average of those at the surrounding FIELD points.

At each point having TYPE equal to AVERAGE, the value at the point in question is set equal to the average of the values at each of the 28 surrounding points (8 in 2D) that has TYPE equal to either, FIELD or IMAGE. At such IMAGE points the location of the corresponding object point is obtained from the IMAGE array, and it is the value at this object point that is used in the average. If WEIGHT="YES" the Jacobian \sqrt{g} , i.e., the cell volume, is calculated at each point from

$$VOL = \sqrt{g} = (r_{\xi 1} \times r_{\xi 2}) \cdot r_{\xi 3}$$

and is multiplied by the value in the formation of the average. Here the field arrays RAD and P are used as dummy arrays to accumulate the average and the number of points, or the volume increments, contributing thereto, for each FIELD point as the field is swept. The field is then swept a second time setting the values to the average.

Image Points

For the setting of values at the IMAGE points, a loop is made over all blocks, each being placed in core by a call to SSD with argument IB. The image points in the block are swept in groups having object points in a single block in order to minimize the accesses to storage, using the LINK array that is set up by subroutine SETIMG, as is described for

subroutine IMGPTS. The value in R at the image point is set to that in R (read into RAD) at the object point. Here IB,IC1,IC2,IC3 identify the image point, while B,C(1),C(2),C(3) identify the corresponding object point. The array R for the object block is read into RAD by calling SSD1 or CCD1.

Other Points

Finally, loops are made over all blocks calling NEUPTS for blocks containing Neumann points, and REFPTS for blocks containing reflective points, to set values at such points. In the 2D mode, i.e., if CMAX(3,B) is 1, R2DPTS is then called for each block to set values on the adjacent layers.

7. SUBROUTINE SETIMP (sets values of control functions at image points)

This routine sets the values of the control functions at image points equal to values determined from the corresponding object and also sets values at other boundary points. Since only the control functions at FIELD points are involved in the generation system, this routine is only necessary because the control functions may be smoothed. If BMAX is 1, either SSDW or CCDW is first called to write the single block on storage. This is necessary because values for the object block are read from storage as discussed below.

Subroutine DEFCON is first called for each block to default the control functions to "NONE" at all non-field points so that a check can be made for unset necessary values after all control functions have been set.

Average Points

In the setting of values at special points designated as AVERAGE points, a loop is made over all blocks, each of which is placed in core by a call to SSD with argument B. The routine AVGCON is then called to set the value.

Image Points

For the setting of values at IMAGE points, a loop is made over all blocks, each of which is placed in core by a call to SSD with argument IB. The image points in the block are swept in groups having object points in a single block in order to minimize the accesses to storage, using the LINK array that is set up by subroutine SETIMG, as is described for subroutine IMGPTS. For each object block, SSD3 or CCD3 is called with argument B to read the coordinate array R, the control function array P, and the type array TYPE into the arrays RAD, SPACE, and ITYPE respectively.

For IMAGE points, account must be taken of the fact that the curvilinear coordinate direction, and even the species thereof, may change on a grid line passing from one block into another, so that the control functions at image points cannot simply be set equal to those at the corresponding object points. This is handled by defining a set of unit tangents to the curvilinear coordinate lines at the image points, these tangents being evaluated by central differences wherever possible and by one-sided differences otherwise. These tangents are placed in the array DRI(3,3), where the first subscript identifies the component and the second the curvilinear direction, i.e.,

$$DRI(J,I) = (r_{\xi^1}^I)_J$$

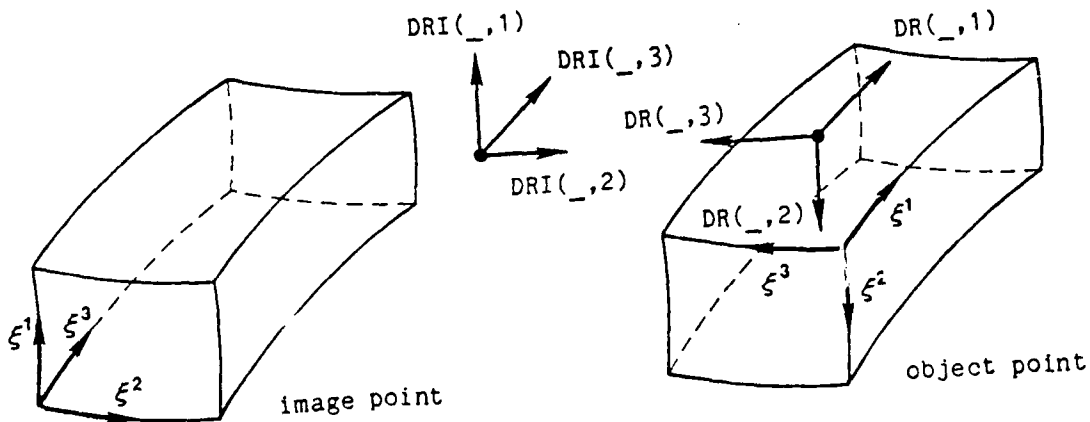
The corresponding object point is located through the IMAGE array in the usual manner, and a similar set of tangents is defined there and placed in the array DR(3,3). The dot products of these two sets of tangents are then formed and placed in the array DOT(3,3) as

$$DOT(I,J) = \sum_{k=1}^3 DRI(k,I) * DR(k,J)$$

Each non-zero element of DOT is then normalized to unit magnitude, retaining the sign. The values of the three control functions at the image point are then set from the values at the object point as

$$P_i^{image} = \sum_{j=1}^3 DOT(i,j) P_j^{object} \quad i=1,2,3$$

As an example, consider the situation diagrammed below:



$$DOT = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$p_1^{image} = -p_2^{object}$$

$$p_2^{image} = -p_3^{object}$$

$$p_3^{image} = p_1^{object}$$

Reflective Points

Values at REFLECT points are set in a loop over all blocks, with each being placed in core by a call to SSD with argument B and the values set by a call to REFCON for each block that contains reflective points.

Other Boundary Points

Finally, a loop is made over all blocks, each being placed in core by a call to SSD with argument B, setting the values at fixed, Neumann, and orthogonal points by calling BONCON. If the integer argument CONCUR is equal to "AVERAGE", subroutine CUTCON is also called to set the values at image points on cuts to averages across the cut. A second pass is then made through the IMAGE and REFLECT points in order to reset those that correspond to the types of points set in BONCON.

8. SUBROUTINE TRANS (transfinite interpolation)

This subroutine determines values in a section identified by BLOCK, START, and END by transfinite interpolation from the section boundaries. The required values on the section boundaries must have been set before this routine is called. (The theory of transfinite interpolation is discussed in Appendix A.)

Arrays

In this routine the section boundary, and the first and cross derivatives thereon, are in the array

$$RB(3,0:1,0:1,0:1,DMRB)$$

where the first subscript is the component, the next three subscripts identify the derivative, e.g., 0,0,0 for these three subscripts indicates the point itself r ; while 1,0,0 indicates the derivative r_{ξ^1} ; 0,1,0 the derivative r_{ξ^2} ; and 1,1,0 the cross-derivative $r_{\xi^1\xi^2}$; etc. The position on the section boundary is given by the last subscript, the value of which comes from the array

$$NDXRB(DIM1, DIM2, DIM3)$$

Thus r and its derivatives at a point with curvilinear coordinates i,j,k are in RB with the last subscript as $NDXRB(i,j,k)$. The array $NDXRB$ is equivalenced with $TYPE$ in the main code. The array $LDXRB$ is the same as $NDXRB$, but is equivalenced with ACC in $MAIN$.

The blending functions, which are needed at every point in the section, are in the array

BLEN(3, 0:1, 0:1, DIM1, DIM2, DIM3)

where the first subscript identifies the interpolation direction and the last three give the position in the section. The second subscript is 0 for Lagrange interpolation and 1 for Hermite, and the third is 0 for the lower end of the interpolation line and 1 for the upper. This array is equivalenced with P in MAIN, and extends over P, SPACE, RAD, and IMAGE in COMMON/FIELD/. The arguments of the blending function are in the array

FACIN(3,DIM1,DIM2,DIM3)

where the first subscript gives the component, and the other three the position in the section. This array is equivalenced with R in MAIN.

For Hermite interpolation, the off-boundary spacing is in the field array

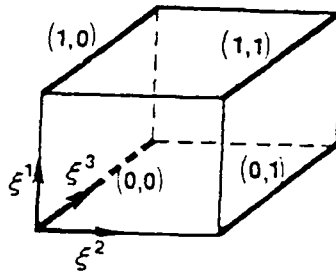
SPAN(SSDB, IS:DIM1, IS:DIM2, IS:DIM3)

which is equivalenced with ACC in MAIN. This spacing may have been read on input; otherwise it will be calculated in this subroutine by interpolation from sides and edges.

For blending functions based on arc length, the array

ARC(3, 0:1, 0:1, DIM)

contains the normalized arc length on the twelve edges of the section. Here the first subscript identifies the curvilinear coordinate (1,2, or 3) which varies along the edge, and the next two subscripts identify one of the four such edges, e.g., with a 3 for the first subscript, the four edges are



The second and third subscripts correspond to the two curvilinear coordinates that are constant on the edge, in cyclic order with the coordinate varying on the edge, with a 0 for the lower side and a 1 for the upper. (Thus in the figure above, the heavily dashed edge has 3,0,0 for the first three subscripts. Values of the second and third subscripts are shown in parentheses for each of the four edges corresponding to a first subscript of 3 in this figure.) The last subscript gives the value of the curvilinear coordinate varying on the edge (e.g. the value of ξ^3 in the figure above). The parameter DIM must conform to the limit

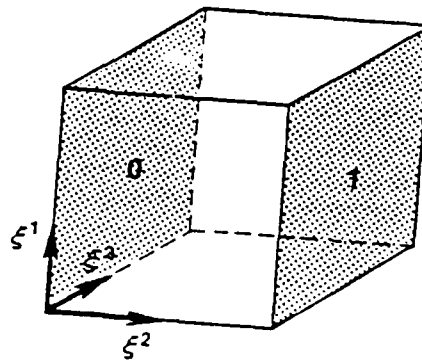
$$\text{DIM} \geq \max(\text{DIM1}, \text{DIM2}, \text{DIM3})$$

and hence is received as the maximum dimensions of the block.

The relative arc length on the sides of the section are in the array

$$\text{ARS}(3, 2, 0:1, \text{DIM}, \text{DIM})$$

where the first subscript identifies the curvilinear coordinate (1, 2, or 3) that is constant on the side, and the third subscript identifies the lower (0) and upper (1) side on which that coordinate is constant, e.g., with 2 for the first subscript the two associated sides are as shown below:



The second subscript corresponds to the two curvilinear coordinates that vary on the side, in cyclic order with the constant coordinate, with 1 corresponding to the first coordinate and 2 corresponding to the other. The last two subscripts give the values of these two coordinates on the side.

This array is equivalenced with P in MAIN, and may span P, SPACE, RAD, IMAGE, ACC, and TYPE in COMMON/FIELD/. This equivalence imposes another limit on DIM since it must be that

$$12 * \text{DIM} * \text{DIM} \leq 15 * \text{DIMT}$$

The equivalence used in this subroutine is merely to save storage, and therefore these limits could be obviated simply by removing the equivalence in MAIN. This limit is checked in MAIN.

Finally the quantity to be interpolated is received in the argument field array

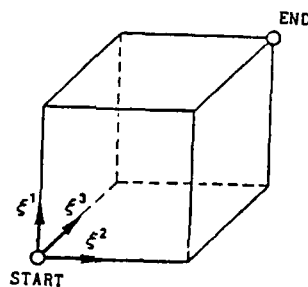
`F(3,SSDB,IS:DIM1,IS:DIM2,IS:DIM3)`

This array will contain either the three Cartesian coordinates or the three control functions.

Setup

Because of the equivalences used, it is necessary to preserve the arrays in COMMON/FIELD/, and, to this end, file 7 is rewound and the arrays F and TYPE are written thereto, followed by a separate rite of R, and then by the arrays R,P,SPACE,RAD,IMAGE, and ACC on another separate write. The interpolation parameters PROJ,PROF,BLEND, and PROTYP are also written on file 7.

The routine then sets PROF(i) to 0, 1, 2, or 3 according to whether the interpolation form is specified as "LAGRANGE", "HERMITE1", "HERMITE2", or "HERMITE" in the direction i, for i=1, 2, and 3. The values in START and END are then interchanged, if necessary, so that all the entries in END are not less than the corresponding entries in START, with the result that the corners defining the section are set as follows:



The counter NRS is set equal to the number of components to be interpolated.

The routine then determines if the section is a volume, surface, or line by checking to see how many entries of END are equal to the corresponding entries of START. Two such equalities indicate a line, one a surface, and none a volume. The dimensionality of the section is placed

in the integer DIMOLD, and the curvilinear directions for which END is not equal to START are recorded in the array DIR(3) as they are found. The integer SAM is one greater than DIMOLD.

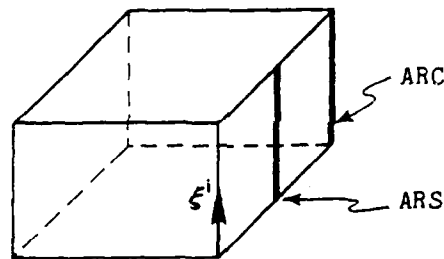
The curvilinear directions in which the interpolation is to be done will have been specified in the array PRODIR(3), which is equivalenced with the array PROJ(3) for transfer to TRANS. If the first entry in PROJ is zero, TRANS defaults the interpolation directions to all that are allowed by the dimensionality of the section. Thus if the section is a line, the first entry of PROJ is reset to the first entry in DIR, the other two entries are set to zero, and PROTYP is set to "FACES". This gives interpolation between the two ends of the line. If the section is a surface then if PROTYP="FACES" the first two entries of PROJ are set to the first two of DIR and the other entry is set to zero. This gives interpolation from all of the surface boundary. If, however, PROTYP is equal to "EDGES" or "CORNERS", then the first entry in PROJ is set to the curvilinear direction off the surface, giving interpolation from the four corners of the surface.

For interpolation for the Cartesian coordinates, after setting LUS(1) to 0, each interpolation direction is checked for completeness of the required values on the section boundaries. If missing values are detected, i.e., if "NONE" is found in the Cartesian coordinate array at a required boundary point, and LUS(2) has not been set to -1 indicating an unresolvable situation, LUS(1) is set to the direction involved and LUS(2) is set to 0 or 1 to indicate whether the deficiency is on the lower or upper boundary in that direction. The routine then returns to the main program for an attempt at interpolation of lesser

dimensionality to supply the needed values. If LUS(2) has been set to -1, and missing values are detected, the code stops, having failed to be able to supply the needed values.

Arc Length

For each curvilinear direction, i , for which the blending functions are to be based on arc length (BLEND(i)="ARC"), the relative arc length distributions are calculated on the four edges, and on the four sides, of a section on which ξ^i varies. These are placed in the arrays ARC and ARS, respectively, as has been described above:



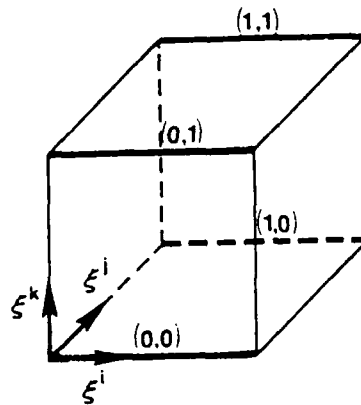
The arc length (actually chord length) increments on the four edges are calculated as

$$ds = |d\mathbf{r}|$$

along the edge in question from the Cartesian coordinates of the points given in the array R, and are placed in the array

DAR(0:1,0:1,DIM)

where the first two subscripts identify the edge, with 0 and 1 corresponding to the lower and upper limits of the two curvilinear coordinates that are cyclic with the one varying on the edge, e.g., with i, j, k cyclic:



The last subscript gives the value of the coordinate varying on the edge, i.e., the position on the edge. These increments are added in separate loops, in the interest of vectorization, to produce the arc length in the array

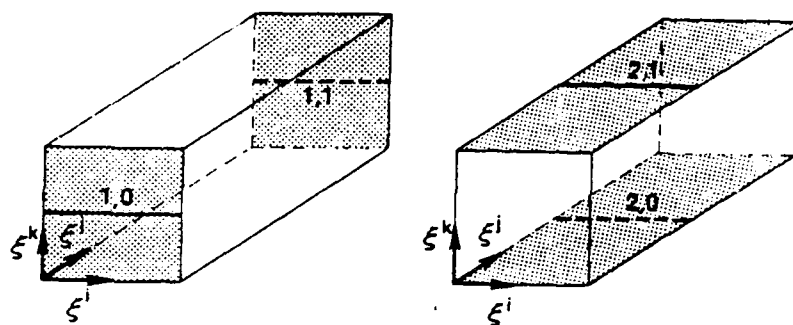
ARC(3,0:1,0:1,DIM)

where the first subscript gives the direction along the edge, 1, and the other three correspond to those of DAR.

If the section is three-dimensional, the arc length on the sides on which ξ^i varies is calculated similarly, with the increments being again placed in DAR, but now with the first subscript identifying the lower and upper side and the second being 0. (DAR is used separately for the two surfaces on which each of the two directions are cyclic with direction 1.) These increments are added and placed in the array

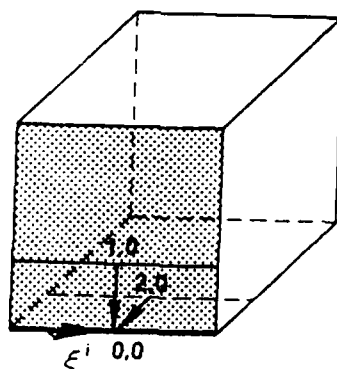
ARS(3,2,0:1,DIM,DIM)

with the first subscript giving the direction along which the arc is taken, i ; the second and third identifying the side by giving the coordinate, in cyclic order with i , that is constant on the side (second) and designating the upper and lower sides for that direction as 0 and 1 (third):

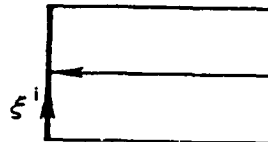


These arc lengths are then examined for zero values, e.g. on polar axes, etc. If the arc length is found to be zero on all four edges, the blending function for the i direction is simply reset to "LINEAR".

For a three-dimensional section, if the arc length on edge 0,0, which lies between side 1,0 and side 2,0:

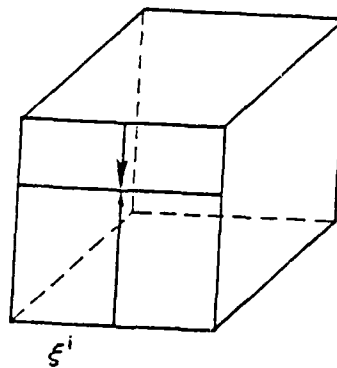


is zero, the arc lengths on this edge are reset to the average between the non-zero values at the first line in the direction of the edge on each of the adjacent sides. For a two-dimensional section, the arc lengths on the edge are reset to the values on the opposite edge:



This procedure is followed for all four edges in the direction i .

For a three-dimensional section, zero arc lengths on the sides are replaced with linearly interpolated values between the two edges bordering the side:



This is done on all four sides on which i varies, after all twelve edges of the section have been done.

Finally all of the arc lengths are converted to relative values by division by the total arc length on each, the total arc having been stored beforehand in the array

TARC(3,0:1,0:1)

for the edges and in the array

TARS(3,2,0:1,DIM)

for the sides. The subscripts in each of these correspond to all but the last subscript in ARC and ARS, respectively. Several separate loops are used here in the interest of vectorization.

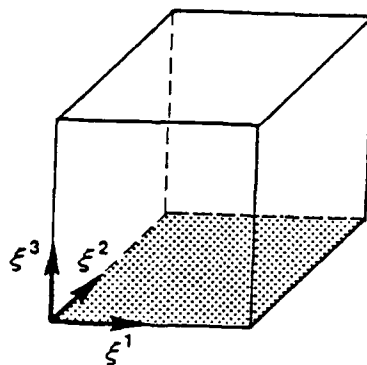
All of the above procedures are repeated for each direction for which BLEND is equal to "ARC".

Index Array

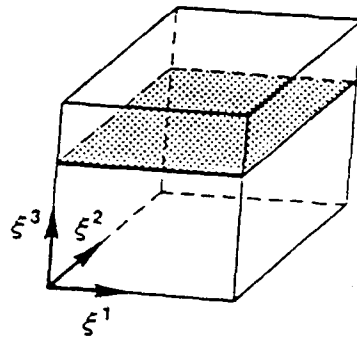
The index array,

NDXRB(DIM1,DIM2,DIM3)

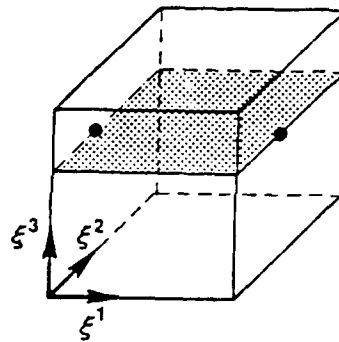
gives the value of the last subscript in the section boundary array, RB, corresponding to a position specified by the three curvilinear coordinates of a point on the boundary. The bottom of the section boundary



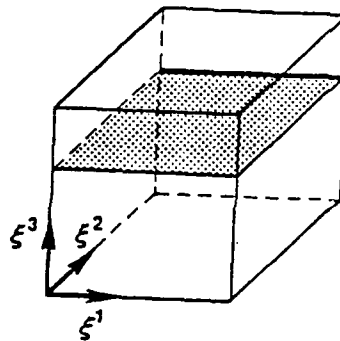
is swept first, with ξ^1 varying faster, placing successively incremented values in NDXRB. In three dimensions, at each successive value of ξ^3 , the line of points on the lower ξ^2 side are added next,



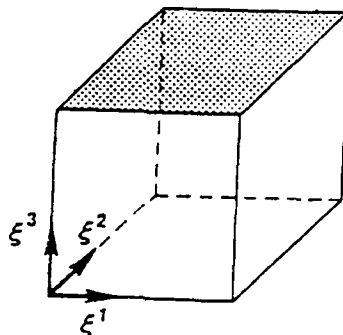
followed by each successive pair of points on the opposite sides on which ξ^1 is constant:



and then by the line of points on the upper ξ^2 side:



Finally the points on the top side are added, again with ξ^1 running faster:



Boundary Array

The section boundary array

`RB(3,0:1,0:1,0:1,DMRB)`

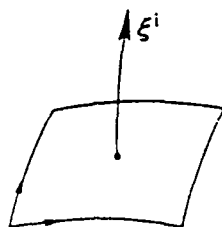
in which the first subscript gives the component, and the next three identify the function and its first and cross derivatives, is filled next. The last subscript identifies the location on the boundary and is that given by the index array `IDXRB`.

If the interpolation is for the control functions, the argument function `F`, containing the quantity to be interpolated, is read into `R` from file 7 after rewinding. Thus the array `R` now contains the quantity to be interpolated in any case.

The points on the section boundary are placed in `RB`, with the second-fourth subscripts as 0,0,0. For each curvilinear direction, i , in which interpolation is to be done, the two sides on which ξ^i is constant are swept, placing the Cartesian coordinates of each point on these

sides from the array R into the array RB. This is done with the curvilinear coordinate on the longer side of the section varying faster, in the interest of vectorization.

If the interpolation is Hermite, the off-surface first derivatives on the sides of the section boundary



are calculated and placed in RB with the second-fourth subscripts therein as 1,0,0 for $r_{\xi 1}$; 0,1,0 for $r_{\xi 2}$; and 0,0,1 for $r_{\xi 3}$. For each curvilinear direction, i , in which interpolation is to be done, the two sides on which ξ^i is constant are swept. At each point on the surface, the tangents, $r_{\xi j}$ and $r_{\xi k}$, and normal, $r_{\xi j} \times r_{\xi k}$, are calculated, with i,j,k cyclic, using central differences in the two directions on the surface. The square of the magnitude of the normal is placed in DN. For blending functions based on arc length, the off-boundary spacing for the Hermite interpolation is (see Appendix A)

$$r_f = \frac{\Delta s}{\Delta a} \bar{n}$$

where Δs is the off-boundary spacing, Δa is the local relative arc length increment, and \underline{n} is the surface unit normal. The off-boundary spacing, Δs , either has been set in the field array SPAN or will be taken as the arc length increment, $A\Delta a$, when A is the local total arc length, in which case

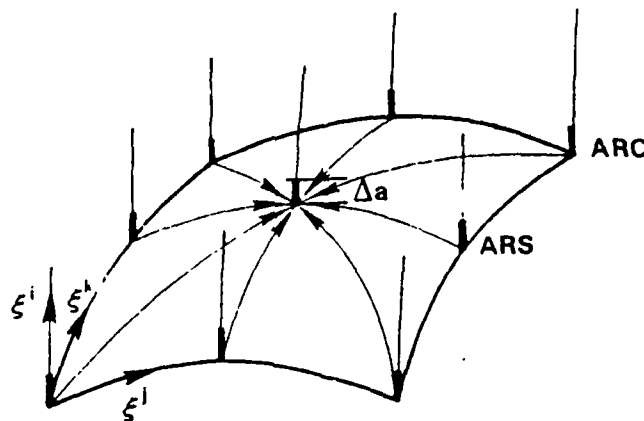
$$\underline{r}_f = A \underline{n}$$

For linear blending functions, the off-boundary spacing, Δs , must have been specified in SPAN and we have

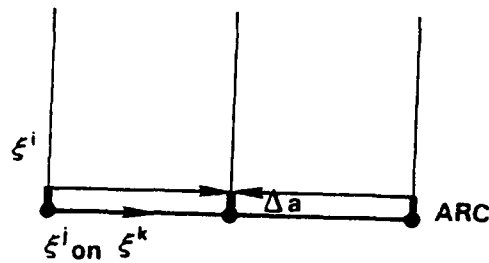
$$\underline{r}_f = I\Delta s \underline{n}$$

where I is the total number of points in the interpolation direction.

In the case of arc blending functions with specified spacing, the local arc length increment, Δa , is determined by transfinite interpolation of one degree less than that for the section, using arc the type of blending functions specified for the two directions on the side. In three dimensions this interpolation then is as shown below:



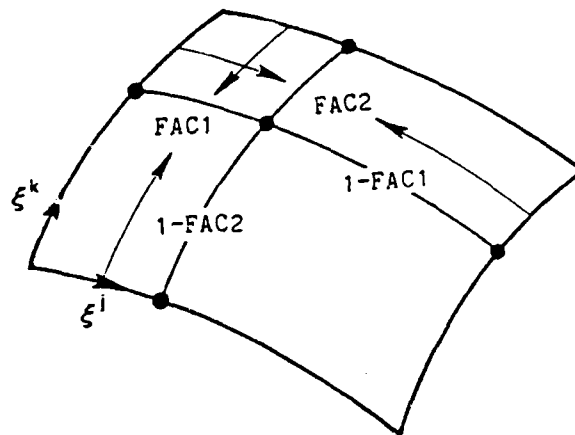
Then in two dimensions we have



The interpolation fractions for linear blending functions on the surface are (Appendix A)

$$FAC1 = \frac{\xi^j - 1}{CMAX(B,J) - 1}, \quad FAC2 = \frac{\xi^k - 1}{CMAX(B,K) - 1}$$

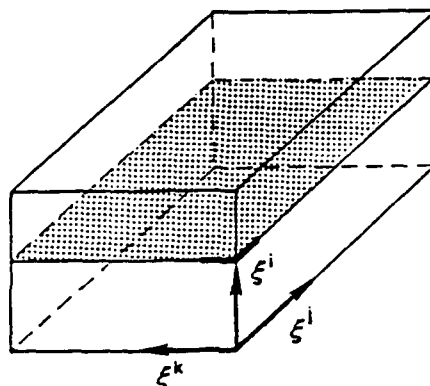
and for arc blending functions on the surface are themselves interpolated from two opposing edges on the surface:



This interpolation for the local relative arc length uses linear blending functions, of course. With arc blending functions but unspecified spacing, the local total arc length, A , is interpolated in the same manner using TARC and TARS.

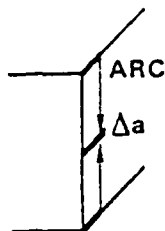
Finally, multiplication of the appropriate quantities by the unit normal gives the first derivatives to be placed in RB with the second-fourth subscripts taken sequentially from the array ID(3), where ID(1)=1 and the other two entries are zero.

Hermite interpolation also requires the first and cross derivatives on the section edges



if the interpolation is more than one-dimensional. These are calculated by sweeping all the edges for which Hermite interpolation is specified in both of the two directions off the edge. For the edge on which ξ^i varies, the tangents to the grid lines intersecting the edge, c_{ξ^j} and c_{ξ^k} , are calculated using one-sided differences in the two directions off the edge.

For each of the two directions off the edge, if the blending functions in the off-edge direction are based on arc length then with specified off-boundary spacing the local arc length increment, Δa , is interpolated linearly between the ends of the edge line:



Otherwise, with unspecified spacing it is the local total arc length, A , that is interpolated, using TARC instead of ARC.

The off-edge first derivatives are formed from the equations given above and are placed in RB with the second-fourth subscripts taken sequentially from ID with $ID(j)=1$ for the derivative in the j direction off the edge, or $ID(k)=1$ for the other, with the other two entries of ID equal to zero in each case.

The cross derivatives on the edge are simply set to zero, these values being entered in RB with $ID(j) = ID(k) = 1$ and the other value zero.

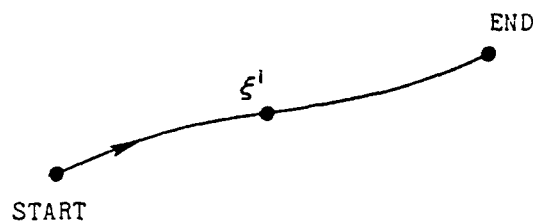
The third derivatives, $\epsilon_{\xi_1 \xi_2 \xi_3}$, at the corners of a three-dimensional section are simply set to zero and are placed in RB with the second-fourth subscripts all equal to 1.

Interpolation Fractions

The interpolation fractions (range 0-1) are set in the entire section for use in calculating the blending functions. These fractions are placed in the array

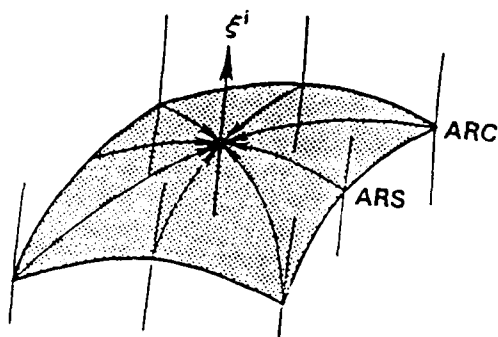
FACIN(3,DIM1,DIM2,DIM3)

where the first subscript identifies the interpolation direction and the other three give the position in the field. This array, and the intermediate array $FI(DIM,3)$, are first filled with zeros. Then for each curvilinear direction in which interpolation is to be done the array FI is set to the local point fraction at each point in the given direction:

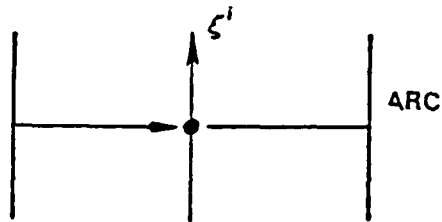


$$FI(\xi^i, i) = \frac{\xi^i - START(i)}{END(i) - START(i)}$$

For each direction, say i , in which interpolation is to be done with blending functions based on arc length, the interpolation fraction is the relative arc length interpolated from the four edges and four sides on which ξ^i varies in a three-dimensional section



or from the two edges on which ξ^i varies in a two-dimensional section:



In either case, the fractions used in this interpolation of the arc lengths are supplied by the array FI, and the interpolated arc lengths are placed in FACIN.

For linear blending functions, the values of the interpolation fractions in FI are simply placed in FACIN directly for each direction in which interpolation is to be done.

Blending Functions

The blending functions for the type of interpolation specified are calculated from the interpolation fractions according to the relations given in Appendix A for each direction in which interpolation is to be done and are placed in the array

BLEN(3,0:1,0:1,DIM1,DIM2,DIM3)

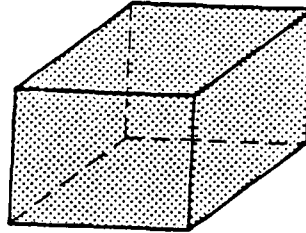
Here the first subscript identifies the direction, and the last three the position in the section. The second subscript is 0 for Lagrange interpolation and 1 for Hermite, and the third is 0 for the former end and 1 for the latter.

Interpolation

The index array NDXRB, which is equivalenced with TYPE, is transferred to the array LDXRB via file 8, after which TYPE is recovered from file 7 after rewinding. If the interpolation is for the radius of curvature, relative arc lengths in the single interpolation direction are transferred from BLEN to ARC, the Cartesian coordinate array is read into RAD from file 7, and subroutine TERPR is called to do the interpolation.

For all other interpolation, the array R, which is to contain the quantity to be interpolated, is set to 0 at all points not designated "FIX", and the interpolation is done at all points in the section not having TYPE="FIX", by calling the appropriate combination of projectors for the type of interpolation specified. The single projectors are invoked by a call to TERP1, the double products by a call to TERP2, and the triple product by TERP3.

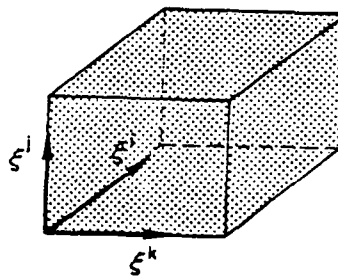
For PROTYP="FACES" the interpolation is from all sides of the section indicated by PROJ. An entry 1 in PROJ causes the pair of sides on which ξ^i is constant to be included. Thus if all three directions are indicated, and the section is a volume, this interpolation is from all six sides



and the combination of projectors is (Appendix A)

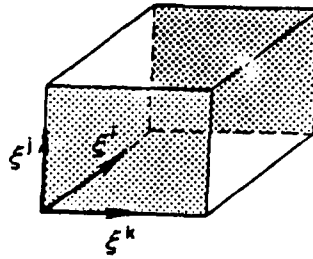
$$P_1 + P_2 + P_3 - P_1P_2 - P_2P_3 - P_3P_1 + P_1P_2P_3$$

while if only the two directions j and k are indicated, or even if all three are indicated but the section is a surface on which ξ^i is constant, the interpolation is from the four sides on which either ξ^j or ξ^k is constant:



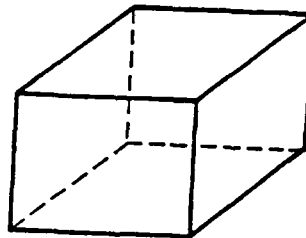
$$P_j + P_k - P_jP_k \quad (i,j,k) \text{ cyclic}$$

With only a single direction i indicated, or in any case if the section is a line on which ξ^i varies, the interpolation is between the two sides on which ξ^i is constant



using only the single projector P_i .

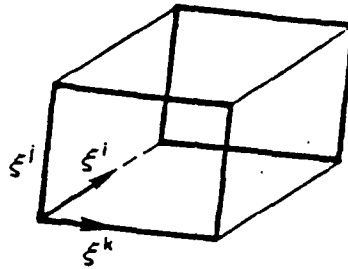
With `PROTYP="EDGES"`, the interpolation is from only those edges of the section indicated by `PROJ`. Here the four edges on which ξ^1 varies are included for an entry i in `PROJ`. With all three directions indicated, and the section a volume, the interpolation is from all twelve edges



using the combination

$$P_1 P_2 + P_2 P_3 + P_3 P_1 - 2P_1 P_2 P_3$$

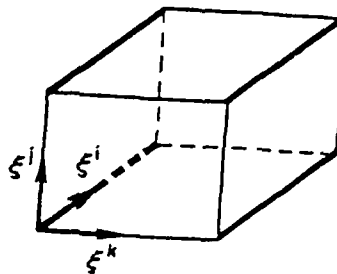
With only the two directions i and j entered in `PROJ`, the interpolation is from the eight edges on which either ξ^i or ξ^j vary



with the combination

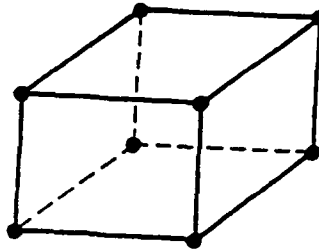
$$P_k P_l + P_m P_n - P_1 P_2 P_3 \quad \begin{matrix} (i,k,l) \\ (j,m,n) \end{matrix} \text{ cyclic}$$

If PROJ contains only the single entry i then the interpolation is from the four edges on which ξ^i varies



using only $P_j P_k$, (i,j,k) cyclic.

If PROTYP="CORNERS" the interpolation is from the eight corners of the section



using $P_1P_2P_3$.

After the interpolation, the interpolated array R is written on file 8, and then R, P, SPACE, RAD, IMAGE, and ACC are recovered from file 7. Finally, the interpolated array on file 8 is read into the appropriate array.

9. SUBROUTINE TERP1 (single projector)

This subroutine receives the interpolation array F, the blending function array BLEN, the index array NDXR, the section boundary array RB, the point type array TYPE, the number of components to be interpolated NRS, and the projection direction N, as arguments and adds the contribution from the single projection to the interpolation array.

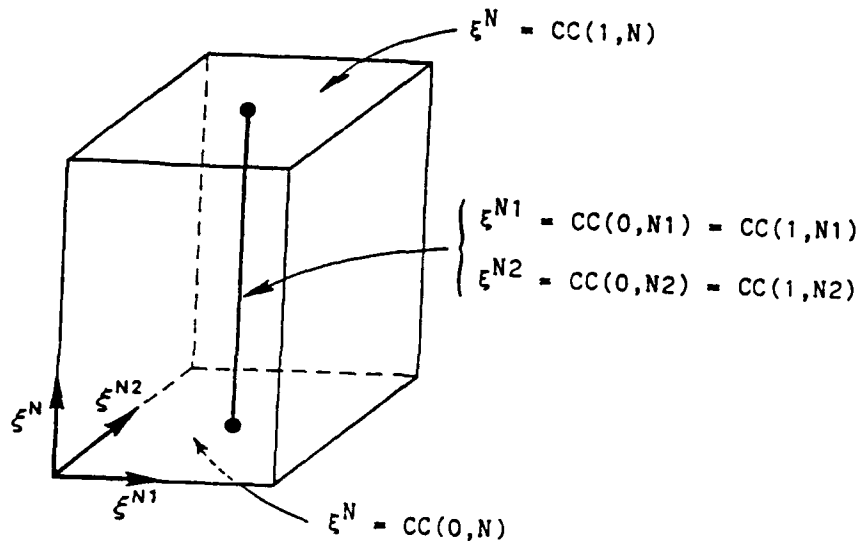
The array LF(0:3) contains the limits for the summation over the forms of blending functions and has the following values, with PROF as its argument:

<u>Interpolation Form</u>	<u>PROF</u>	<u>LF</u>
LAGRANGE	0	0
HERMITE1	1	1
HERMITE2	2	1
HERMITE	3	1

The array LE(2,0:1,0:3) contains the limits for the summations over the ends of the interpolation range, with the form of the interpolation, i.e., PROF, as the last subscript, the summation index over the blending function forms as the second subscript, and the limit identified by the first (0 for the lower and 1 for the upper). The values are as follows:

<u>Interpolation Form</u>	<u>PROF</u>	<u>Summation Limits for 1st Blending Function</u>	<u>Summation Limits for 2d Blending Function</u>
LAGRANGE	0	0,1	---
HERMITE1	1	0,1	0,0
HERMITE2	2	0,1	1,1
HERMITE	3	0,1	0,1

Here the integer array CC(0:1,3) contains the values of the curvilinear coordinate that is constant on each of the two sides of the section from which the projection is made, with the curvilinear direction given by the second subscript and the side identified by the first (0 for lower and 1 for upper). The other four entries in CC are the values of the curvilinear coordinates that are constant on the projection line. Thus, with N, N1, N2, cyclic,



Two entries of this array are set from START and END, which are received through COMMON/SECTN/, while the other four vary with the projection line.

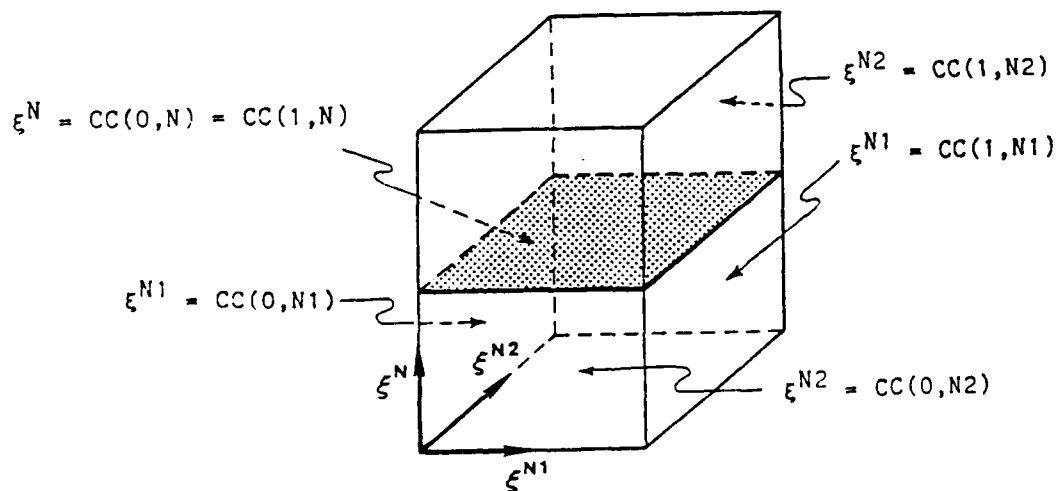
Each projection line in the section is swept, whereupon for each point not classified as "FIX", the two summations given in Appendix A (Eq. A-18) are performed for each component to be interpolated, as specified by the array RS(3) in COMMON/SECTN/, adding the result to the interpolation array, F.

10. SUBROUTINE TERP2 (double projector product)

This subroutine receives the interpolation array F, the blending function array BLEN, the index array NDXRB, the section boundary array RB, the point type array TYPE, the number of components to be interpolated NRS, and the two directions in the product N1, N2, as arguments and adds the contribution from the double projector product to the

interpolation array. In this routine the integer array ACY(3,3) gives the direction other than that specified by two unequal arguments, e.g., ACY(1,2)=ACY(2,1)=3. The arrays LF and LE serve as described for TERP1.

Here the integer array CC(0:1,3) contains the value of the curvilinear coordinate that is constant on each of the four sides of the section from which the projection is made, with the curvilinear direction given by the second subscript and the side identified by the first (0 for lower and 1 for upper). The remaining two entries in CC are both set to the value of the curvilinear coordinate that is constant on the projection plane defined by the product specified. Thus, with N, N1, N2 cyclic,



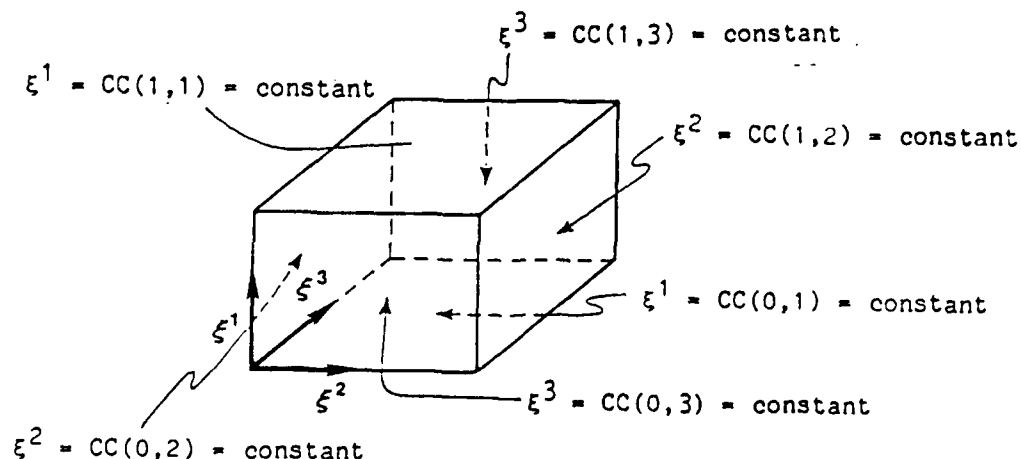
Four entries of this array are set from START and END, which are received through COMMON/SECTN/, while the other two vary with the projection planes.

Each projection plane in the section is swept, whereupon for each point not classified as "FIX", the four summations given in Appendix A (Eq. A-19) are performed for each component to be interpolated, as specified by the arrays RS(3) in COMMON/SECTN/, adding the result to the interpolation array, F.

11. SUBROUTINE TERP3 (triple projection product)

This subroutine receives the interpolation array F, the blending function array BLEN, the index array NDXRB, the section boundary array RB, the point type array TYPE, the number of components to be interpolated NRS, and a sign, SN, as arguments and adds the contribution from the triple projector product to the interpolation array. (This contribution is negative if SN is so.) The arrays LF and LE serve as described for TERP1.

Here the integer array CC(0:1,3) contains the value of the curvilinear coordinate that is constant on each of the six sides of the section, with the curvilinear direction given by the second subscript and the side identified by the first (0 for lower and 1 for upper):



This array is filled from START and END, which are received through COMMON/SECTN/.

For each point in the section not classified as "FIX", the six summations given in Appendix A (Eq. A-20) are performed for each component to be interpolated as specified by the array RS(3) in COMMON/SECTN/, adding the result to the interpolation array, F.

12. SUBROUTINE TERPR

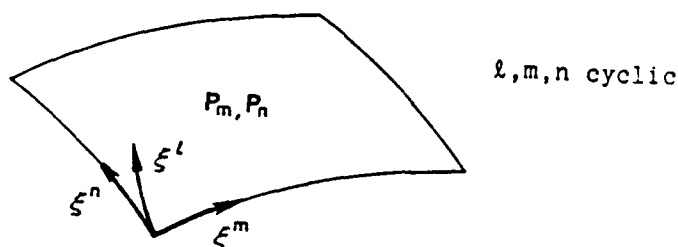
This subroutine calculates the radius of curvature for control functions corresponding to CONTYP="RADIUS" as in Appendix G. The code notation is as follows, with reference to the appendix:

RAD	:	ρ
RAD1,RAD2	:	ρ_1, ρ_2
TAL	:	A
RADT	:	$\rho_1 + A, \rho_2 + A$
DEL	:	δ
SS	:	s
SO	:	s_0

13. SUBROUTINE CONSURF (surface control functions, including both arc length and curvature contributions)

This subroutine evaluates the control functions on a surface section identified by BLOCK, START, and END, with one pair of corresponding entries in START and END being equal. These functions include both the arc length and curvature contributions, and thus can be interpolated directly into the interior.

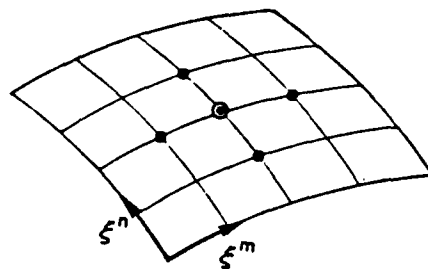
The two control functions evaluated on the surface are those in the two curvilinear directions on the surface.



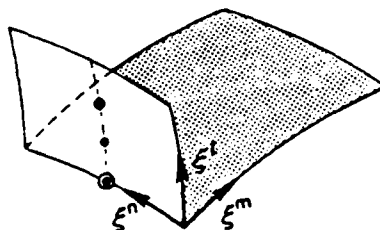
These are given by the equation (Eq. C-24 of Appendix C)

$$P_m = - \frac{1}{g_{ll}} \left(r_{\xi^m} - \frac{g_{mn}}{g_{nn}} r_{\xi^n} \right) \cdot r_{\xi^l \xi^l} - \left(\frac{r_{\xi^m} - \frac{g_{mn}}{g_{nn}} r_{\xi^n}}{g_{mm}g_{nn} - g_{mn}^2} \right) \cdot (g_{nn} r_{\xi^m \xi^m} + g_{mm} r_{\xi^n \xi^n} - 2g_{mn} r_{\xi^m \xi^n})$$

with l, m, n , cyclic, and an analogous equation with m and n interchanged. The development of this equation is given in Appendix C. Here l is the curvilinear direction off the surface, and m and n are the two directions on the surface. The first term (involving the second derivative off the surface) is the contribution from the curvature of the lines intersecting the surface, while the other terms involve only derivatives on the surface and represent the arc length contribution. Derivatives on the surface can, of course, be evaluated from the point distribution on the surface, so that the arc length contribution can be evaluated locally:



The curvature contribution must, however, be interpolated from the edges of the surface where the off-surface derivatives can be evaluated on the intersecting surfaces:

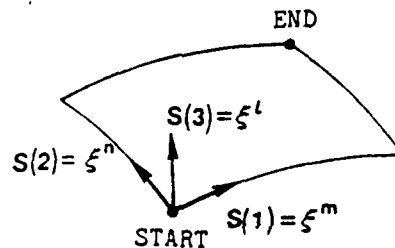


The routine first determines a scale below which values are taken to be zero as $|\Delta \zeta|^2 \times 10^{-16}$, where $\Delta \zeta$ is the diagonal of the section extending from START to END.

This limit is used to check the metric elements g_{mm} and g_{nn} which appear as denominators. A zero for these elements can only mean the direction involved is meaningless because of a reduction to two dimensions, and hence the denominator is reset to 10^{20} to eliminate the term involved. The form of the limit follows from the fact that $g_{mm} = |\zeta_m|^2$, etc., meaning that cells with sides smaller than 10^{-8} times the diagonal

of the section are considered to be the result of a reduction to 2D. This limit would have to be changed if such small cells really do exist. The square of this limit is also used on the denominator $g_{mn}g_{nn}-g_{mn}^2$.

The code then determines which curvilinear coordinate is constant on the surface, by checking for equality of corresponding entries in START and END, and makes this direction (l) the third entry in the array S(3). The other two entries in S are set to the other two directions (m and n), in cyclic order with that in S(3):



The entire surface is then swept, calculating the control functions at interior points on the section as follows. First all the coordinate derivatives, i.e., first, second, and cross, in the two curvilinear directions on the surface are calculated by central differences and placed in the array DR(3,0:3,0:3):

$$DR(k,1,0) = (r_{\xi^1})_k$$

$$DR(k,1,j) = (r_{\xi^1 \xi^j})_k$$

with i and j assuming both of the curvilinear directions on the surface, i.e., the first two entries in S , and the component index, k , assuming the values 1,2,3. Then all the dot products of these first and second derivatives in the two directions on the surface are placed in the array DOT (3,3,3) where

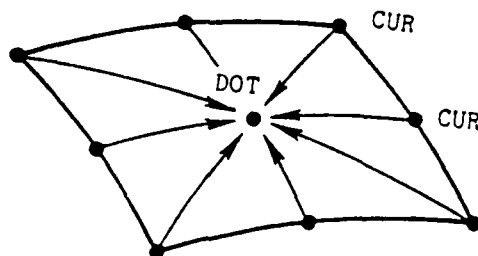
$$\text{DOT}(k,i,j) = r_{\xi} S(k) \cdot r_{\xi} S(i) r_{\xi} S(j)$$

with k,i,j each assuming both of the values 1,2. The surface elements of the covariant metric tensor are placed in the array G(3,3):

$$G(i,j) = g_{S(i),S(j)} = r_{\xi} S(i) \cdot r_{\xi} S(j)$$

again with i and j assuming both of the values 1,2.

The contribution to the surface control functions from the curvature of the coordinate line intersecting the surface is then determined by calling subroutine OFF (with OFFTYP="SURFACE") at the four points on the section boundary that are opposite the point where the functions are being evaluated, and also at the four corners of the section:



In the calls to OFF, the array CC(3) contains the curvilinear coordinates of the point on the section boundary at which the curvative contribution is to be returned. The dot products

$$\frac{r_{\xi^m} \cdot r_{\xi^l \xi^l}}{g_{ll}} \quad \text{and} \quad \frac{r_{\xi^n} \cdot r_{\xi^l \xi^l}}{g_{ll}}$$

are returned in CUR(1) and CUR(2) for the two control functions through COMMON/CONOFF/. The dot products for curvature contribution at the point of evaluation of the control functions are then interpolated from the values at these eight points on the section boundary, by transfinite interpolation and are placed in DOT(1,3,3) with i assuming the two values 1,2.

The complete surface control functions for the two curvilinear directions on the surface are then calculated by adding the curvature contribution, using DOT(1,3,3) with i=1,2, to the arc length contribution calculated using DOT(k,i,j) with k,i,j each assuming the values 1,2, implementing the equations given at the beginning of this section. In the code notation, m=S(1), n=S(2) and l=S(3), with the metric elements written here as

$$g_{mn} = g_{S(1),(2)} = G(1,2), \text{ etc., and}$$

$$g_{ll} = g_{S(3),S(3)} = G(3,3)$$

and the dot products as

$$\begin{aligned} r_{\xi^m} \cdot r_{\xi^{mn}} &= r_{\xi^{S(1)}} \cdot r_{\xi^{S(1)S(2)}} \\ &= \text{DOT}(1,1,2), \text{ etc.} \end{aligned}$$

and

$$\frac{1}{g_{\xi\xi}} (r_{\xi m} \cdot r_{\xi \ell \ell}) = (r_{\xi(1)} \cdot r_{\xi S(3)S(3)}) / |r_{\xi(3)}|^2$$

$$= \text{DOT}(1,3,3), \text{ etc.}$$

The former dot products are the arc length contributions, while the latter are the curvature contributions.

The equations for the control functions then use the following notation:

$$\text{DENOM} = g_{mm} g_{nn} - g_{mn}^2$$

$$\text{DENOM1} = g_{mm}$$

$$\text{DENOM2} = g_{nn}$$

$$\text{RHS1} = r_{\xi m} \cdot \frac{(g_{nn} r_{\xi m \xi m} + g_{mm} r_{\xi n \xi n} - 2g_{mn} r_{\xi m \xi n})}{g_{mm} g_{nn} - g_{mn}^2} + r_{\xi \ell \ell}$$

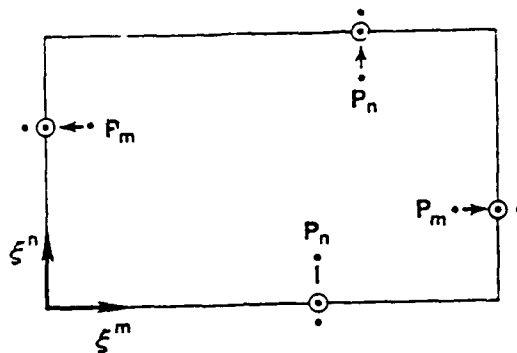
$$\text{RHS2} = r_{\xi n} \cdot \frac{(g_{nn} r_{\xi m \xi m} + g_{mm} r_{\xi n \xi n} - 2g_{mn} r_{\xi m \xi n})}{g_{mm} g_{nn} - g_{mn}^2} + r_{\xi \ell \ell}$$

$$p_m = - \frac{\text{RHS1} * g_{nn} - \text{RHS2} * g_{mn}}{g_{nn}}$$

$$p_n = - \frac{\text{RHS2} * g_{mm} - \text{RHS1} * g_{mn}}{g_{mm}}$$

This routine sweeps the entire section, including the edges, calculating the three elements of the control functions. However, if no Cartesian coordinates have set on the surrounding layers, the control

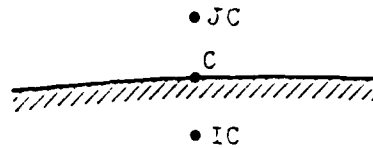
function P_m will not be evaluated properly on an edge on which ξ^m is constant, since this requires a derivative across the edge. Therefore, each of the four edges is examined and values of P_m at "FIX", "NEUMANN", "ORTHO" and "AVERAGE" points on the edges on which ξ^m is constant are extrapolated from adjacent points if the value of the first component of the Cartesian coordinate array R is equal to the default value, "NONE", at the adjacent point outside the edge.



The extrapolation for edge values is also done at "IMAGE" points. This has no effect at points imaged to "FIELD" points, since then R will have a real value outside the edge and good control functions will have been calculated there. However, points imaged to non-field points such as "FIX", "ORTHO", etc. must have extrapolated control functions since none are calculated there.

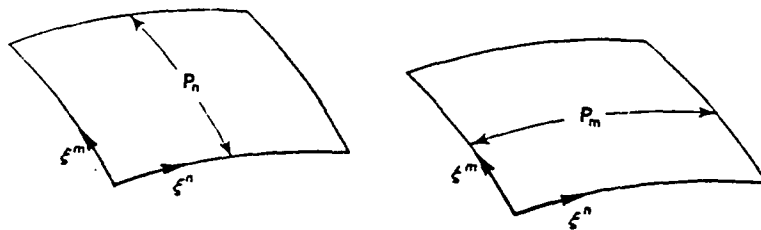
If the coordinates have been set at the adjacent exterior point, but the coordinates at either this point or at the adjacent interior point are the same as those at the point on the edge, then the edge is part of a surface that corresponds to a polar axis, and consequently the control functions there are set to zero.

The code uses the notation $C(3)$ for the point on the edge, $IC(3)$ for the adjacent exterior point, and $JC(13)$ for the adjacent interior point:



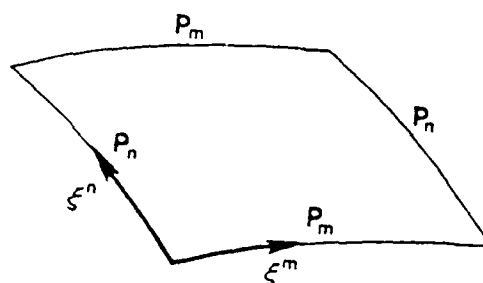
The physical distances (squared) from the edge point to the exterior and interior point, respectively, are RAX and SAX , and a value of either of these less than the square of the inverse of the radius of curvature scale, indicates a polar axis.

If the surface control functions are to be smoothed ($SMOCON="YES"$), the control function in each curvilinear direction on the surface is smoothed one-dimensionally in the other direction on the surface, replacing the function at each point by the average of the value there and the average of the values at the points on either side:



The $S(3)$ component of P is used as the dummy as the $S(1)$ and $S(2)$ components are smoothed on the surface section.

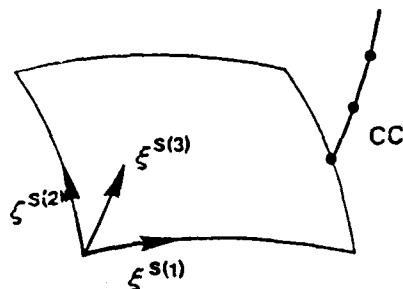
Finally, subroutine CONLINE is called on each of the four edges of the section to calculate the remaining control functions at "FIX", "NEUMANN", and "ORTHOG" points there as follows:



This involves redefining START and END to define the edge to CONLINE, and hence the section values of these quantities are saved and are reset before returning.

14. SUBROUTINE OFF (off-surface derivatives for surface control functions)

This subroutine calculates the dot products for curvature contribution to the surface control functions. It receives the three curvilinear coordinates of a point on an edge of the surface section in the array CC(3) through COMMON/SECTN/. The order in which these coordinates appear in CC is given by the permutation of 1,2,3 appearing in the array S(3), transferred through the same common. The first two entries in S are the two curvilinear directions on the surface, in cyclic order with the third, while the third is the direction off the surface:



The coefficients for difference expressions for the first and second derivatives are determined as follows:

$$f_o^{(i)} = \frac{f_{SU(i)} - f_{SL(i)}}{|SU(i)| + |SL(i)|}$$

$$f_o^{(ii)} = f_{S1(i)} - 2f_{S2(i)} + f_{S3(i)}$$

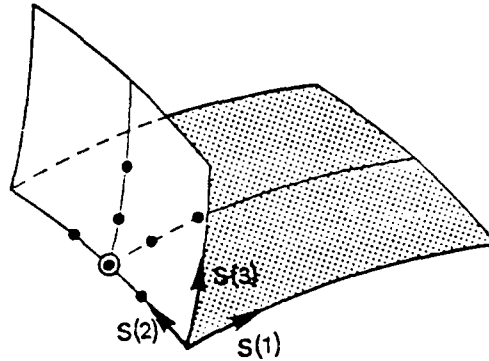
where the superscripts indicate derivatives in the i direction, and where the subscripts are

	SU	SL	S1	S2	S3
control	1	-1	1	0	1
forward	1	0	0	1	2
backward	0	-1	0	-1	-2

These subscripts are set up as three-entry arrays, and appropriate values are entered for derivatives in each curvilinear direction.

The code checks the values of each of the curvilinear coordinates in CC for equality with unity or the corresponding coordinate limit in CMAX, using forward differences in the former case, backward in the latter, and central otherwise, to form the three first and second coor-

dinate derivatives, $r_{\xi i}$ and $r_{\xi i \xi i}$, at the point. Central differences are also used across an edge when the points on the adjacent surrounding layer are "IMAGE" points.



Here these derivatives are placed in the arrays D1(3,3) and D2(3,3):

$$D1(k, S(i)) = (r_{\xi S(i)})_k$$

$$D2(k, S(i)) = (r_{\xi S(i)} \xi S(i))_k$$

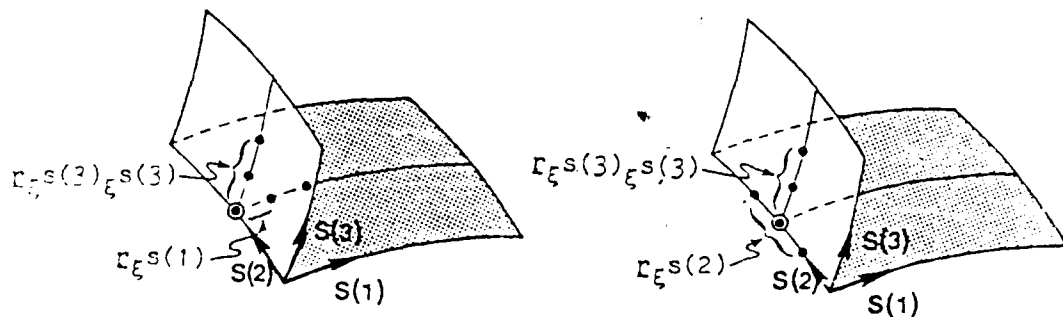
and the diagonal covariant metric elements are placed in G(3):

$$G(S(i)) = |r_{\xi S(i)}|^2$$

In these, i , and the component index, k , assume the values 1,2,3. The dot products for the curvature contribution to the two surface control functions are then calculated and placed in the array CUR(3). In 3D, with OFFTYP="SURFACE", the calculation is

$$CUR(i) = \frac{r_{\xi S(i)} \cdot r_{\xi S(3)} \xi S(3)}{|r_{\xi S(3)}|^2}$$

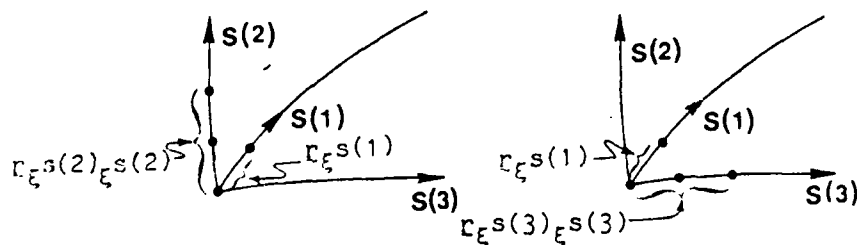
where i assumes the two values 1,2. Recall that the curvilinear direction $S(3)$ is off the surface, while $S(1)$ and (2) are the two directions on the surface:



In 2D the curvature contribution is calculated on the ends of a line and becomes

$$CUR(i) = \frac{r_{\xi} S(1) \cdot r_{\xi} S(i)_{\xi} S(1)}{|r_{\xi} S(i)|^2}$$

Here i assumes the two values 2,3. In this case $S(1)$ is the curvilinear direction along the line, $S(2)$ and $S(3)$ being the other two directions:



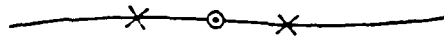
15. SUBROUTINE CONLINE (line control function, including both length and curvature contributions)

This subroutine evaluates the control function at "FIX", "NEUMAN", and "ORTHOG" points on a line section identified by BLOCK, START, and END, with two pairs of corresponding entries in START and END being equal. This function includes both the arc length and curvature contributions.

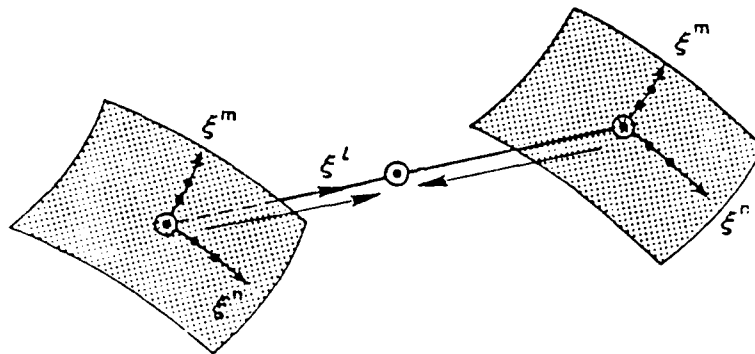
The single control function evaluated on the line is given by (Eq. C-20 of Appendix C)

$$P_{\ell} = - \frac{r_{\xi \ell} \cdot r_{\xi \ell \ell}}{g_{\ell \ell}} - r_{\xi \ell} \cdot \left(\frac{r_{\xi^m \xi^m}}{g_{mm}} + \frac{r_{\xi^n \xi^n}}{g_{nn}} \right)$$

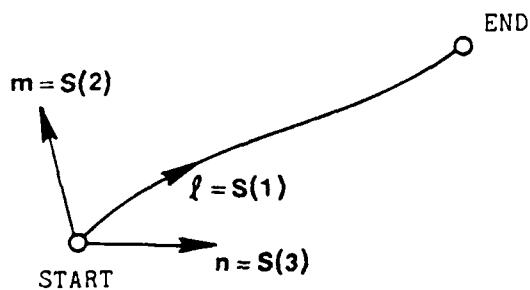
with ℓ the curvilinear direction along the line and m and n cyclic with ℓ . The development of this equation is given in Appendix C. The first term here is the arc length contribution and is evaluated locally from the point distribution on the line.



The other term is the curvature contribution and requires interpolation from the ends of the line where the two off-line derivatives can be evaluated from the intersecting surface.



The routine first scales for zero denominators as in CONSURF, and then determines which curvilinear coordinate varies on the line, by checking for equality of corresponding entries in START and END, and places this direction (l) as the first entry in the array S . The other two entries in S are set to the other two directions (m and n), in cyclic order with the first:



The control function in the direction along the line is calculated at each point between the ends as follows. First the coordinate derivatives along the line are calculated using central differences and are placed in the array DR(3,0:3,0:3):

$$DR(k,i,0) = (r_{\xi^i})_k$$

$$DR(k,i,i) = (r_{\xi^i \xi^i})_k$$

with $i=S(1)$, and the component index, k , assuming the values 1,2,3. The dot products of these first and second derivatives along the line are then placed in the array DOT as

$$DOT(1,1,1) = r_{\xi^i} \cdot r_{\xi^i \xi^i}$$

and the diagonal covariant metric element g_{ii} is placed in the array G:

$$G(1,1) = r_{\xi^1} \cdot r_{\xi^1}$$

Subroutine OFF is then called with OFFTYP="LINE" to calculate the dot products

$$\frac{r_{\xi^l} \cdot r_{\xi^m \xi^m}}{g_{mm}} \quad \text{and} \quad \frac{r_{\xi^l} \cdot r_{\xi^n \xi^n}}{g_{nn}}$$

for the curvature contributions on each end of the line, returning these quantities as CUR(2) and CUR(3). The values of these dot products at the point where the control function is being evaluated are then interpolated between the two end values and are placed in DOT(1,2,2) and DOT(1,3,3).

The complete line control function for the curvilinear direction along the line is then calculated by adding the curvature contributions, using DOT(1,2,2) and DOT(1,3,3), to that from the arc length, using DOT(1,1,1), implementing the equation given at the beginning of this section. In the code, $l=S(1)$, $m=S(2)$, and $n=S(3)$, the matrix elements here are

$$g_{ll} = G(1,1)$$

$$g_{mm} = G(2,2)$$

$$g_{nn} = G(3,3)$$

and the dot products are

$$r_{\xi l} \cdot r_{\xi ll} = \text{DOT}(1,1,1)$$

$$\frac{1}{g_{mm}} (r_{\xi l} \cdot r_{\xi mm}) = \text{DOT}(1,2,2), \text{ etc.}$$

Finally the control functions at the ends of the line are interpolated from the adjacent values for end points that are "FIX", "AVERAGE", "NEUMANN", or "ORTHOG" and where no Cartesian coordinates have been set outside the end. As in CONSURF, the control functions are set to zero when the end point has $r_{\xi l} = 0$ on either side, i.e., is on a polar axis.

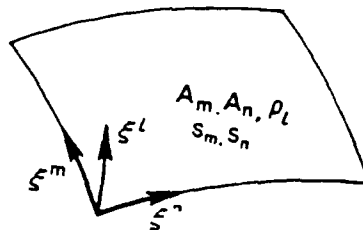
The extrapolation for edge values is also done at "IMAGE" points. This has no effect at points imaged to "FIELD" points, since then R will have a real value outside the edge and good control functions will

have been calculated there. However, points imaged to non-field points such as "FIX", "ORTHOG", etc. must have extrapolated control functions since none are calculated there.

16. SUBROUTINE CONSURR (surface control functions with separate arc length and curvature contributions)

This routine separately evaluates an arc length contribution and the elements of a curvature contribution to the control functions on a surface section identified by BLOCK, START, and END, with one pair of corresponding entries in START and END being equal.

These control functions are formed in three elements, each of which is interpolated separately into the field. Thus on the surface section on which ξ^m and ξ^n vary (on which ξ^l is constant),



this routine calculates the arc length contributions, A_m and A_n , the spacings, s_m and s_n , the radius of curvature of the surface, ρ_l , from the equations (Eq. C-6 and C-20 of Appendix C)

$$A_m = - \frac{\underline{r}_{\xi^m} \cdot \underline{r}_{\xi^m \xi^m}}{g_{mm}}, \quad A_n = - \frac{\underline{r}_{\xi^n} \cdot \underline{r}_{\xi^n \xi^n}}{g_{nn}}$$

$$s_m = \sqrt{g_{mm}}, \quad s_n = \sqrt{g_{nn}}$$

$$\rho_l = - \left[\frac{\underline{r}_{\xi^m} \cdot \underline{r}_{\xi^m \xi^m}}{g_{mm}} + \frac{\underline{r}_{\xi^n} \cdot \underline{r}_{\xi^n \xi^n}}{g_{nn}} \right]^{-1}$$

where \underline{n} is the unit normal to the surface, calculated from

$$\underline{n} = \frac{\underline{r}_{\xi^m} \times \underline{r}_{\xi^n}}{|\underline{r}_{\xi^m} \times \underline{r}_{\xi^n}|}$$

Appendix C contains the development of these terms. The arc length contributions A_m and A_n , are placed as the m and n components of the field array P , the spacings s_m and s_n as the m and n components of the field array $SPACE$, and the radius of curvature ρ_l as the l component of the field array RAD .

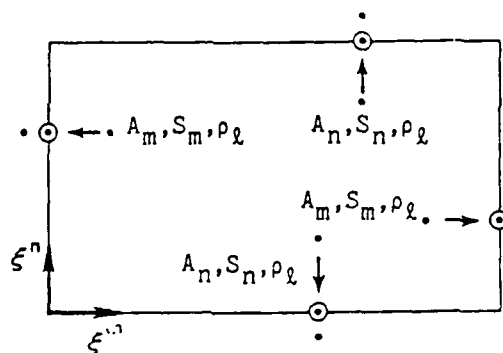
The code determines the two surface directions, m and n , and the off-surface direction, l , as in `CONSURF`, again placing these as the three entries in S . The first and second derivatives on the surface, the associated metric elements, and the necessary dot products, are also determined as in `CONSURF`. The unit normal is placed in the array `NOR(3)`, and the dot products involved in ρ are placed in `DOT(3,3,3)` as

$$DOT(3,1,1) = \underline{n} \cdot \underline{r}_{\xi^m \xi^m}$$

$$DOT(3,2,2) = \underline{n} \cdot \underline{r}_{\xi^n \xi^n}$$

Normals with magnitudes less than $(\sqrt{g_{11}} + \sqrt{g_{22}}) \times 10^{-8}$ are considered to be zero, and a zero value for the radius of curvature is assigned. If the bracket, i.e., the curvature, in the equation for the radius of curvature is less than the inverse of the radius of curvature scale, `RSCAL`, and the normal is not zero, the surface is taken to be locally flat and the radius of curvature is set to the radius of the curvature scale, i.e., essentially infinity.

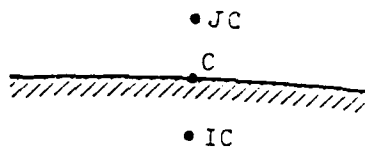
This routine sweeps the entire section, including the edges, calculating the three elements of the control functions. However, if no Cartesian coordinates have set on the surrounding layers, the elements A_m , s_m , and ρ_l will not be evaluated properly on an edge on which ξ^m is constant, since these require a derivative across the edge. Therefore, each of the four edges is examined and values of the elements A_m , s_m , and ρ_l at "FIX", "NEUMANN", "ORTHO" and "AVERAGE" points on the edges on which ξ^m is constant are extrapolated from adjacent points if the value of the first component of the Cartesian coordinate array R is equal to the default value, "NONE", at the adjacent point outside the edge.



The extrapolation for edge values is also done at "IMAGE" points. This has no effect at points imaged to "FIELD" points, since then R will have a real value outside the edge and good control functions will have been calculated there. However, points imaged to non-field points such as "FIX", "ORTHO", etc. must have extrapolated control functions since none are calculated there.

If the coordinates have been set at the adjacent exterior point, but the coordinates at either this point or at the adjacent interior point are the same as those at the point on the edge, then the edge is part of a surface that correspond to a polar axis, and consequently all these elements of the control functions are set to zero.

The code uses the notation $C(3)$ for the point on the edge, $IC(3)$ for the adjacent exterior point, and $JC(13)$ for the adjacent interior point:



The physical distances (squared) from the edge point to the exterior and interior point, respectively, are RAX and SAX , and a value of either of these less than the square of the inverse of the radius of curvature scale, indicates a polar axis.

If smoothing on the surface is called for, the arc length contribution to the control function in P , and the arc length spacing in $SPACE$, in each curvilinear direction on the surface are both smoothed along the other direction on the surface as in $CONSURF$. The radius of curvature, however, is smoothed in both directions on the surface, replacing the

value at each point by the average of the value there and the average of the values at the four adjacent points. The as yet unused l component of the P array is used for the dummy in the smoothing.

Finally CONLINR is called to calculate the elements of the remaining control functions on the four edges of the surface, analogous to the procedure in CONSURF.

17. SUBROUTINE CONLINR (Line control functions with separate arc length and curvature contributions)

This routine separately evaluates an arc length contribution and the components of a curvature contribution to the control function on a line section identified by BLOCK, START, and END, with two pairs of corresponding entries in START and END being equal, analogous to the surface evaluation in CONSURR.

The line is identified as in CONLINE, with the direction along the line (l) being placed in $S(3)$, and the other two directions (m and n) in $S(1)$ and $S(2)$, in cyclic order with $S(3)$. The first and second derivatives, the required dot products, and metric coefficients along the line, are also calculated as in CONLINE.

In the 3D case only the arc length contribution (Eq. C-6 and C-20 of Appendix C)

$$A_l = - \frac{\vec{r}_{\xi l} \cdot \vec{r}_{\xi l}}{g_{ll}}$$

is evaluated, this being placed in P as in CONSURR. In 2D the spacing

$$s_l = \sqrt{g_{ll}}$$

in SPACE, and the radius of curvature

$$\rho_l = - \frac{\underline{n} \cdot \underline{r}_{\xi^l \xi^l}^{-1}}{g_{ll}}$$

in RAD, are also calculated on the line.

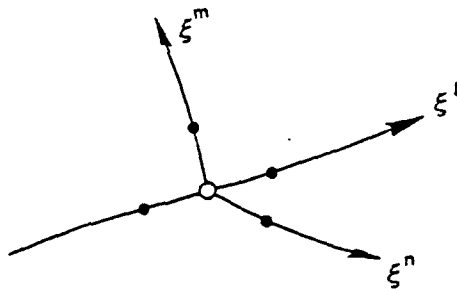
There are two normals to the line here, one obtained as

$$\underline{n}^{(1)} = \frac{\underline{r}_{\xi^n} \times \underline{r}_{\xi^l}}{|\underline{r}_{\xi^n} \times \underline{r}_{\xi^l}|}$$

and the other as

$$\underline{n}^{(2)} = \frac{\underline{r}_{\xi^l} \times \underline{r}_{\xi^m}}{|\underline{r}_{\xi^l} \times \underline{r}_{\xi^m}|}$$

Here central differences are used along the line and also in the two directions off the line if possible, with one-sided differences being used otherwise.



The dot products needed for ρ_l are then placed as

$$\text{DOT}(2,1,1) = \underline{n}^{(1)} \cdot \underline{r}_{\xi^l \xi^l}$$

$$\text{DOT}(3,1,1) = \underline{n}^{(2)} \cdot \underline{r}_{\xi^l \xi^l}$$

Two radii of curvature are then calculated, using $\eta^{(1)}$ and $\eta^{(2)}$, respectively, and are placed as the m and n components in RAD.

Values at "AVERAGE", "FIX", "NEUMANN", and "ORTHOG" points on the ends of the line are extrapolated from adjacent values when no Cartesian coordinates have been set outside the ends, or are set to zero at points on a polar axis, as in CONLINE.

The extrapolation for edge values is also done at "IMAGE" points. This has no effect at points imaged to "FIELD" points, since then R will have been calculated there. However, points imaged to non-field points such as "FIX", "ORTHOG", etc. must have extrapolated control functions since none are calculated there.

If smoothing is called for, only the two radii of curvature in RAD are smoothed.

18. SUBROUTINE CONINT (control functions from algebraic grid)

This routine evaluates the control functions at all "FIELD" points from the initial algebraic grid. The three components of the elliptic grid generation system (Appendix B) provide a set of three equations,

$$\sum_{k=1}^3 g^{kk}(\underline{r}_k)_l P_k = - \sum_{i=1}^3 \sum_{j=1}^3 g^{ij}(\underline{r}_{ij})_l \quad l=1,2,3$$

that can be solved simultaneously at each point for the three control functions, P_k ($k=1,2,3$). The derivatives here are represented by central differences. This option is exercised if CONTYP="INITIAL" and produces control functions which will reproduce the algebraic grid from the elliptic system solution in a single iteration if central differences are specified. Thus evaluation of the control functions in this manner would be of trivial interest except when these control functions are smoothed before being used in the elliptic generation system.

A variation of this evaluation procedure is to set the off-diagonal metric elements to zero in the evaluation of the control functions from the algebraic grid, as if the grid were orthogonal. This produces a grid from the elliptic system that has the same qualitative line distribution as the algebraic grid but is more orthogonal. This variation is invoked with CONTYP="ORTHO".

A third variation sets only the two off-diagonal metric elements having one index the same as that of the control function to zero. This amounts to relaxing the assumption of orthogonality in the curvilinear coordinate surface associated with each control function.

19. SUBROUTINE VOLSYS (elliptic grid generation system)

This subroutine performs one point SOR iteration of the elliptic grid generation equations, calculating new values of the Cartesian coordinates at all "FIELD" points. The SOR sweep direction is alternated so that the iteration is symmetric point SOR.

All of the first and second coordinate derivatives are first calculated, using second-order central differences, and are placed in the array DR(3,0:3,0:3) as follows:

$$(r_{\xi^1})_{\ell} = DR(\ell, 1, 0) = \frac{1}{2} [r(\xi^1 + 1) - r(\xi^1 - 1)]_{\ell}$$

$$(r_{\xi^i \xi^j})_{\ell} - 2\delta_{ij} r_{\ell} = DR(\ell, i, j) = [r(\xi^i + 1) + r(\xi^i - 1)]_{\ell} \quad \text{for } i=j$$

or

$$= \frac{1}{4} [r(\xi^i + 1, \xi^j + 1) + r(\xi^i - 1, \xi^j - 1)]$$

$$- r(\xi^i + 1, \xi^j - 1) - r(\xi^i - 1, \xi^j + 1)]_{\ell} \quad \text{for } i \neq j$$

where δ_{ij} is the Kroneker delta and only the arguments that vary have been shown. Next the covariant metric elements are calculated and placed in the array G(3,3):

$$g_{ij} = r_{\xi^i} \cdot r_{\xi^j} = G(i, j)$$

The products of the contravariant metric elements with the square of the Jacobian are placed in the array GG(3,3):

$$gg^{il} = g_{jm} g_{kn} - g_{jn} g_{km} = GG(i, l) \quad \begin{array}{l} (i, j, k) \text{ cyclic} \\ (l, m, n) \text{ cyclic} \end{array}$$

(In 2D, g^{33} is set to zero.)

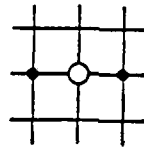
The elliptic grid generation system is

$$\sum_{i=1}^3 \sum_{j=1}^3 g^{ij} r_{\xi^i \xi^j} + \sum_{k=1}^3 g^{kk} P_k r_{\xi^k} = 0$$

where the P_k are the control functions. (This equation is discussed in Appendix B.)

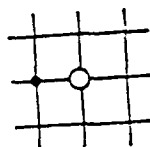
With central differences, the first derivatives in this equation are simply

$$r_{\xi^k} = DR(_, k, 0)$$

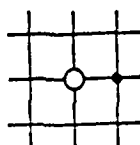


where the first subscript gives the component (1, 2, or 3). Provision is also made for one-sided differences dependent on the sign of the control function P_k (backward for $P_k < 0$ and forward for $P_k > 0$):

$$(r_{\xi^k})_l = [r(\xi^k) - r(\xi^k - 1)]_l \quad P_k < 0$$



$$(r_{\xi k})_l = [r(\xi^{k+1}) - r(\xi^k)]_l \quad P_k > 0$$



The product $P_k r_{\xi k}$ can then be represented as

$$P_k r_{\xi k} = P_k DR(_, k, 0) + A |P_k| \left[\frac{1}{2} DR(_, k, k) - r \right]$$

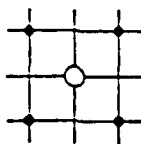
with $A=0$ for central differences and $A=1$ for one-sided. The switching coefficient, A , can be variable:

$$A = 1 - \frac{2}{\max(|P_k|, 2)}$$

giving central differences for $|P_k| \leq 2$, and completely one-sided differences for $|P_k| \rightarrow \infty$. This insures that a one-dimensional overlap will not occur (cf. Appendix I).

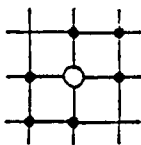
Also, with central differences the cross derivatives are

$$r_{\xi^i \xi^j} = DR(_, i, j)$$



while the skewed cross derivatives are given by

$$\begin{aligned} (r_{\xi^i \xi^j})_k = \frac{1}{2} [& r(\xi^{i+1}, \xi^{j+1}) + r(\xi^{i-1}, \xi^{j-1}) - r(\xi^{i+1}, \xi^j) \\ & - r(\xi^{i-1}, \xi^j) - r(\xi^i, \xi^{j+1}) - r(\xi^i, \xi^{j-1}) \\ & + 2r(\xi^i, \xi^j)] \quad g^{ij} > 0 \end{aligned}$$



and

$$(r_{\xi^i \xi^j})_k = \frac{1}{2} [r(\xi^{i+1}, \xi^{j-1}) + r(\xi^{i-1}, \xi^{j+1}) - r(\xi^{i+1}, \xi^j)]$$

$$\begin{aligned}
& - r(\xi^{i-1}, \xi^j) - r(\xi^i, \xi^{j+1}) - r(\xi^i, \xi^{j-1}) \\
& + 2r(\xi^i, \xi^j)] \quad g^{ij} < 0
\end{aligned}$$



These can be incorporated into one form for the product $g^{ij} r_{\xi^i \xi^j}$ as

$$\begin{aligned}
g^{ij} r_{\xi^i \xi^j} &= g^{ij} DR(_, i, j) \\
&+ B |g^{ij}| \frac{1}{2} [DC(_, i, j) \\
&- DR(_, i, i) - DR(_, j, j) + r]_\ell
\end{aligned}$$

with $B=0$ for central differences and $B=1$ for skewed, and with

$$\begin{aligned}
DC(\ell, i, j) &= \frac{1}{2} [r(\xi^{i+1}, \xi^{j+1}) + r(\xi^{i-1}, \xi^{j-1}) \\
&+ r(\xi^{i+1}, \xi^{j-1}) + r(\xi^{i-1}, \xi^{j+1})]_\ell
\end{aligned}$$

for $i \neq j$.

The generation equation can then be written as

$$\begin{aligned}
& [\sum_k GG(k, k) (2 + A |P_k|) - B \sum_{\substack{i, j \\ (i \neq j)}} |GG(i, j)|] r \\
&= \sum_k GG(k, k) [(1 + \frac{A}{2} |P_k|) DR(_, k, k) + P_k DR(_, k, 0)] \\
&+ \sum_{\substack{i, j \\ (i \neq j)}} \{ GG(i, j) DR(_, i, j) \}
\end{aligned}$$

$$+ \frac{B}{2} |GG(i,j)| [DC(_,i,j) - DR(_,i,i) - DR(_,j,j)] \}$$

where $GG(i,j)$ is g^{ij} times the square of the Jacobian. This is implemented as

$$x_l = \frac{SUM1 + SUM2}{SUM}$$

where

$$SUM = \sum_k GG(k,k) (2+A|P_k|) - B \sum_{\substack{i,j \\ (i \neq j)}} |GG(i,j)|$$

and

$$SUM1 = \sum_k GG(k,k) [P_k DR(_,k,0) + \frac{A}{2} |P_k| DR(_,k,k)]$$

$$SUM2 = \sum_k GG(k,k) DR(_,k,k) + \sum_{\substack{i,j \\ (i \neq j)}} \{ GG(i,j) DR(_,i,j) \\ + \frac{B}{2} |GG(i,j)| [DC(_,i,j) - DR(_,i,i) - DR(_,j,j)] \}$$

The switching coefficient A is obtained from the relation

$$A = A_1 + A_2 \min(1, \frac{1}{2} |P_k|)$$

where A_1 and A_2 assume the following values depending on the differences:

	<u>centered</u>	<u>one-sided</u>	<u>variable</u>
A_1	0	1	0
A_2	0	0	1

with the code notation AFIRS1 and AFIRS2 for A_1 and A_2 . Also used is ACROS2 for $2B$, the two arising from the symmetric sum of which B is the coefficient.

The values of the Cartesian coordinates given by this equation are taken as intermediate values, and the acceleration process yields the new values at the current iteration as

$$x_l^{\text{new}} = \omega x_l + (1-\omega)x_l^{\text{old}}$$

where the acceleration parameter, ω , is in the field array ACC.

The iteration error norms $E(3, \text{BMAX})$ are the maximum values over all "FIELD" points of the absolute change in the values of the Cartesian coordinates between iterations:

$$E(l, \text{BLOCK}) = \max |x_l^{\text{new}} - x_l^{\text{old}}|$$

The location of this maximum is also noted and placed in $EC(i, \text{BLOCK}, l)$, $i=1,2,3$.

20. SUBROUTINE OPTACC (variable field of optimum acceleration parameters)

This subroutine evaluates an acceleration parameter at each "FIELD" point for the point SOR iterative solution of the elliptic generation system, placing these values in the field array ACC. First the coordinate derivatives, $r_{\xi i}$, and the metric elements, g_{ij} and g^{ij} , are evaluated as in subroutine VOLSYS.

The partial differential equation that is solved to generate the grid is Eq.(B-11) of Appendix B. In this equation the coefficient of the central value of r is

$$a_0 = \sum_k g^{kk} (2+A|P_k|) - B \sum_{\substack{i,j \\ (i \neq j)}} |g^{ij}|$$

and the coefficients of r at ξ^{i+1} and at ξ^{i-1} on the right hand side are

$$a_{\pm}^i = g^{ii} \left[\left(1 + \frac{A}{2} |P_i|\right) \pm \frac{1}{2} P_i \right] - B (|g^{ij}| + |g^{ik}|) \quad i=1,2,3$$

with (i,j,k) cyclic in this last equation. Also the product $a_+^i a_-^i$ is given by

$$a_+^i a_-^i = \left[g^{ii} \left(1 + \frac{A}{2} |P_i|\right) - B (|g^{ij}| + |g^{ik}|) \right]^2 - \frac{1}{4} (g^{ii} P_i)^2$$

From Ref. (12), for the difference equation

$$\sum_i a_{\pm}^i x(\xi^{i+1}) + a_-^i x(\xi^{i-1}) - a_0 x = 0$$

(where only the functional dependence on the argument that varies from the central value is shown), a field of optimum acceleration parameters for accelerated point Gauss-Seidel iteration can be calculated as follows:

$$WR : \sqrt{\omega^2 + 4\omega}$$

$$WPAR : \omega \mp \sqrt{\omega^2 + 4\omega}$$

The coefficients A and B that determine the form of the first and cross derivatives are as used in VOLSYS.

The Jacobi eigenvalue (i.e., for point Jacobi iteration) with the largest magnitude is given by

$$\mu = \mu_r + i\mu_i = \frac{2}{a_0} \sum_i \sqrt{a_+^i a_-^i} \cos \frac{1}{N^i + 1}$$

where ξ^i is the number of points in the i direction involved in the solution. (The i on the left side here is, of course, $\sqrt{-1}$, and μ_i is the imaginary part of μ , while on the right, i indicates one of the three directions).

The locally optimum acceleration parameter, ω , is then given by

$$\omega = -\frac{1}{2} \left[\omega \mp \sqrt{\omega^2 + 4\omega} \right]$$

with the upper sign for $\alpha^2 > \beta$ and the lower for $\alpha^2 < \beta$ where

$$\alpha = \mu_r^2 + \mu_i^2$$

$$\beta = \mu_r^2 - \mu_i^2$$

$$a = \alpha^2 - \beta^2$$

$$b = \alpha^2 - \beta$$

$$c = \sqrt{a + b^2}$$

$$\bar{\omega} = \frac{1}{\alpha^2 b} \{ [(3b+c)(c-b)^{1/3} - (3b-c)(c+b)^{1/3}] a^{1/3} + \alpha^2 + 3\beta^2 - 4\alpha^2 \beta \}$$

This is implemented in the code with the following notation:

$$GG(i,j) : g^{ij} \quad \text{SUM} = 2 \sum_k g^{kk}$$

$$AP_i : |P_i|$$

$$AGP_i : 1 + \frac{A}{2} |P_i|$$

$$GGP_i : g^{ii} (1 + \frac{A}{2} |P_i|) |P_i|$$

$$A_{00} : a_0$$

$$A_i : [g^{ii} (1 + \frac{A}{2} |P_i|) - B (|g^{ij}| + |g^{ik}|)]^2$$

$$B_i : \frac{1}{4} (g^{ii} P_i)^2$$

$$\text{Thus } \sqrt{a_+^i a_-^i} = A_i - B_i$$

$$S_i : 2 \sqrt{a_+^i a_-^i} \cos \frac{1}{N^i + 1}$$

COS1 : $2 \cos \frac{1}{N^1 + 1}$ with $N^1 = \text{CMAX}(i, \text{block})$
 PREAL : μ_r
 PIMAG : μ_i
 AU : α
 BU : β
 AL : a
 BL : b
 ABR : c
 ROOT1 : $[a(c-b)]^{1/3}$
 ROOT2 : $[a(c+b)]^{1/3}$
 WBAR : $\bar{\omega}$

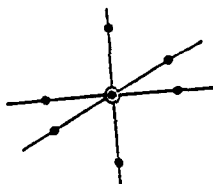
If the control functions are being iteratively adjusted for boundary orthogonality (Section I-C10), it is necessary to limit the acceleration parameters as discussed in Appendix H. Therefore ω , in this case, is limited to the minimum of the value calculated above and the value given by

$$\frac{\sum_k g^{kk}}{\max_k g^{kk}}$$

The routine concludes with the determination and printing of the extreme values of the acceleration parameters over the field.

21. SUBROUTINE SMOOTH (smoothes Cartesian coordinates or control functions)

This routine smoothes either the Cartesian coordinates or the control functions at each "FIELD" point. In either case the field array to be smoothed is first written into a dummy array received as the second argument. At each "FIELD" point the values are then replaced by the average of the values at all the adjacent points in each curvilinear direction in which smoothing has been called for through entries of these directions in the array SMODIR(3). Up to six adjacent points can thus be involved:

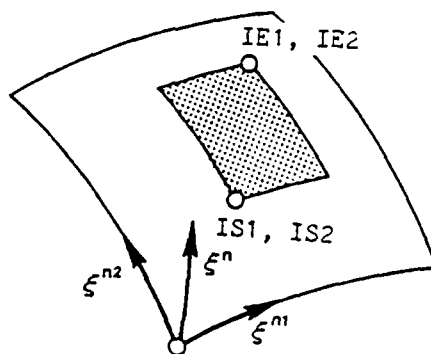


The argument array G is first written on file 8, after rewinding, to free this array to be used as a dummy. This array is recovered before exiting the subroutine by rewinding and reading file 8.

22. SUBROUTINE CONSETU (set up for boundary orthogonality through iterative adjustment of control functions)

The procedure for iterative adjustment of the control functions during the solution of the elliptic grid generation equation is discussed in detail in Appendix D. This subroutine sets up the off-boundary spacing.

The routine examines the block for sections designated for this type of boundary orthogonality, these being counted in ORTNUM. (This and the other arrays involved for the section specification are discussed in Section II-C6.) For such sections, the curvilinear coordinate ξ^n that is constant on the section is obtained as the absolute value of ORTNOR. The value of this coordinate, placed in C(N), and the section limits are obtained from ORTBON:



(N,N1,N2 cyclic)

In the notation of Appendix D, N corresponds to the off-surface direction, l , while N1 and N2 correspond to the two directions, m and n , on the surface.

The section is then swept, with the local curvilinear coordinate values on the section in C(N1) and C(N2), and those relative to the section corner in J1 and J2. The index of the local point in the section array, RO, is obtained from the index function NDXRO with J1 and J2 as arguments. At each "ORTHO" point on the section the following is done:

The first derivatives on the surface are calculated by central differences for the m and n directions and one-sided differences for the l direction, and are placed in DR:

$$DR(_,1,0) : r_{\xi^m}$$

$$DR(_,2,0) : r_{\xi^n}$$

$$DR(_,3,0) : r_{\xi^l}$$

The off-surface spacing, $\sqrt{g_{ll}}$, (GOOR in the code notation) is set from the field array SPAN if an input specification was made; otherwise this spacing is set to $\underline{n} \cdot r_{\xi^l}$ where \underline{n} is the unit normal

$$\underline{n} = \frac{r_{\xi^m} \times r_{\xi^n}}{|r_{\xi^m} \times r_{\xi^n}|}$$

and r_{ξ^l} is evaluated from the initial algebraic grid, and is placed in SPAN.

The code notation for the vectors of Appendix D is as follows:

$$\underline{a} : RO(_,0,_,_)$$

$$\underline{B}_1 : RO _,1,_,_)$$

B_2 : RO(__, 2, __, __)

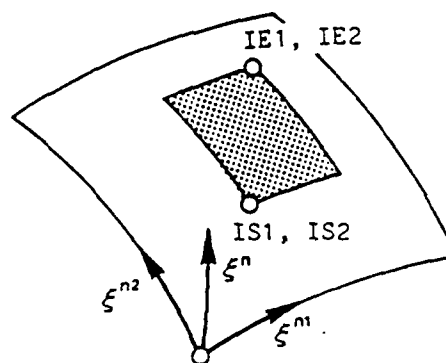
C : RO(__, 3, __, __)

A : V

23. SUBROUTINE CONSETE (iterative adjustment of control functions for boundary orthogonality)

This subroutine changes the control functions throughout the field in order to achieve boundary orthogonality on specified boundary sections, as discussed in Appendix D. The routine first saves the field array RAD on file 7 and then zeroes this array.

The routine examines each block for sections designated for this type of boundary orthogonality, these being counted in ORTNUM. (This and the other arrays involved in the section specification are discussed in Section II-C6.) For such sections, the curvilinear coordinate ξ^n that is constant on the section is obtained as the absolute value of ORTNOR. The value of this coordinate, placed in C(N), and the section limits are obtained from ORTBON:



(N, N1, N2 cyclic)

In the notation of Appendix D, N corresponds to the off-surface direction, ℓ , while N1 and N2 correspond to the two directions, m and n, on the surface.

The section is then swept, with the local curvilinear coordinate values on the section in C(N1) and C(N2) and those relative to the section corner in J1 and J2. The index of the local point in the section array RO is obtained from the index functions NDXRO with J1 and J2 as arguments. At each "ORTHOG" point on the section the following is done:

First the curvilinear coordinate line from the ORTHOG point to the corresponding point on the opposite boundary is swept, calculating the arc length, ARC, along this line.

This line is then swept again. At each point on the line the first and second derivatives on the intersected coordinate surface are calculated by central differences and are placed in DR:

$$DR(_,1,0) : r_{\xi m}$$

$$DR(_,2,0) : r_{\xi n}$$

$$DR(_,1,1) : r_{\xi m \xi m}$$

$$DR(_,2,2) : r_{\xi n \xi n}$$

$$DR(_,1,2) : r_{\xi m \xi n}$$

The first derivative along the line is calculated by a backward difference:

$$DR(_,3,0) : r_{\xi \ell}$$

The metric elements then are calculated and placed as

$$G(1,1) : g_{mm} = |r_{\xi m}|^2$$

$$G(2,2) : g_{nn} = |r_{\xi n}|^2$$

$$G(1,2) : g_{mn} = r_{\xi m} \cdot r_{\xi n}$$

At the first point on the line after the ORTHOG point, the off-boundary spacing is set from the SPAN array, and a hyperbolic sine distribution of arc length is determined along the line having this initial spacing. This distribution is given by

$$s(\xi) = \frac{\sinh(\delta \frac{\xi-1}{I-1})}{\sinh \delta} S$$

where ξ is ξ^k , I is the total number of points on the line, and S is the total arc length. The parameter δ is determined by the initial spacing from

$$\frac{\sinh \delta}{\delta} = \frac{S}{(I-1)*SPAN}$$

This nonlinear equation is solved by Newton iteration. In the less frequent case where the specified off-boundary spacing in SPAN is greater than the average spacing $S/(I-1)$, the hyperbolic functions are replaced by the corresponding circular functions.

At each point on the line the local off-surface spacing, $\sqrt{g_{ll}}$, is calculated from this distribution and is placed in GOOR. The quantities a , B_1 , B_2 , and C of Appendix D are then calculated.

The code rotation for the vectors of Appendix D is as follows:

$$\underline{a} : RO(_,0,_,_)$$

$$\underline{B}_1 : RO(_,1,_,_)$$

$$\underline{B}_2 : RO(_,2,_,_)$$

$$\underline{C} : RO(_,3,_,_)$$

$$\underline{A} : V$$

At each point, the off-surface derivative $r_{\xi^l \xi^l}$ is evaluated as

$$r_{\xi^l \xi^l} = (r_{\xi^l})_{\xi^l} = r_{\xi^l} - \underline{n} \sqrt{g_{ll}}$$

with $r_{\xi^l \xi^l}$ evaluated as a one-sided difference toward the boundary. Here \underline{n} is the local normal to the ξ^l -constant surface:

$$\underline{n} = \frac{r_{\xi^m} \times r_{\xi^n}}{|r_{\xi^m} \times r_{\xi^n}|}$$

The vector \underline{A} of Appendix D is calculated, and the increments in the three control functions at the point are then calculated from

$$\Delta P_m = -\underline{B}_1 \cdot \underline{A} - P_m$$

$$\Delta P_n = -\underline{B}_2 \cdot \underline{A} - P_n$$

$$\Delta P_l = -\underline{C} \cdot \underline{A} - P_l$$

and are placed in DP(N1), DP(N2), and DP(N), respectively. The present values of the control functions are in the field array P. The magnitude of the control function increments is limited to 2.0.

These control function increments are then attenuated by a factor, DFAC, and are added to the array RAD at the point on the line. The attenuation factor, which decreases from 1 at the ORTHOG point to 0 at the other end of the line, is calculated from the ratio of the local spacing to the initial spacing. The attenuation factor is the minimum of this ratio and its inverse, with the additional stipulation that the attenuation factor does not increase along the line.

After the increments from all "ORTHOG" points have been radiated into the field and accumulated, the extreme values of the increments over the field are determined and printed. The control functions at all points in the field are then changed by the increments in R, after which this array is restored by reading from file 7, but attenuated by multiplication by

$$DFAC = 0.1 \left\{ \frac{(\sqrt{g_{ll}})_0}{s} \left[1 - \frac{(\sqrt{g_{ll}})_0}{S} \right] \right\}^{IEXP}$$

where $(\sqrt{g_{ll}})_0$ is the specified off-boundary spacing, and s is the local arc length and where IEXP is 2 in 2D and 3 in 3D. The form of this formula is chosen so that with the same spacing specified at each of two opposing boundaries, the value of DFAC will not exceed 0.1. Thus at the first point off the boundary we have, with $s_0 = (\sqrt{g_{ll}})_0$,

$$0.1 \left[\frac{s_0}{1-s_0} (1 - s_0) \right] + 0.1 \left[\frac{s_0}{s_0} (1 - s_0) \right] = 0.1$$

where the first term arises from the near boundary, and the second from the far. The attenuation factor is multiplied by the input factor

CONFAC (defaulted to 1.0). The attenuation factor is decreased by 1/2 at ORTHOG points that are adjacent to non-ORTHOG points.

24. SUBROUTINE SPLSUR (surface spline)

This routine splines a surface section as described in Appendix E. The section is received in the argument array SUR, with second subscript 0, and the two dimensions of the section are received as the arguments N1 and N2.

The surface spline array SUR is dimensioned SUR(3,0:5,N1,N2) where the last two subscripts are the surface dimensions. The second subscript corresponds to the spline coefficients as follows:

0	r
1	r_{ξ}
2	r_{η}
3	$r_{\xi\eta}$
4	$r_{\xi\xi}$
5	$r_{\eta\eta}$

The curve spline array CUR is dimensioned as in the revision of Section II-F25. The notation is as follows:

SUR (0, ---)	:	r	} surface
SUR (1, ---)	:	r_{ξ}	
SUR (2, ---)	:	r_{η}	
SUR (3, ---)	:	$r_{\xi\eta}$	

CUR (0, ---)	:	r	}	curve
CUR (1, ---)	:	r_{ξ}		
CUR (3, ---)	:	$r_{\xi\xi}$		

The routine places the Cartesian coordinates of the points on the surface in the array CUR, with a zero for the second subscript, and calls SPLCUR to spline each curve of each of the two families on the surface, with the argument "QUAD" for quadratic ends, i.e., constant curvature. The first derivative is returned in CUR with 1 as the second subscript and is transferred to SUR with 1 or 2 as the second subscript, corresponding to r_{ξ} or r_{η} as above. These first derivatives are then splined by calling SPLCUR with the argument "SPECIF" for specified slopes at the ends to generate the cross-derivative as discussed in Appendix E.

25. SUBROUTINE SPLCUR (curve spline)

This routine splines a curve as described in Appendix E. The curve is received as the argument CUR with the second subscript 0, the number of points thereon as NP, the number of components as NC, and the type of ends as TYPE. The ends will have zero curvature (natural spline) for TYPE equal to "NATURAL", constant curvature for "QUAD", or specified slope for "SPECIF". In the latter case the specified slope is received in CUR (2, ---) at the ends.

The curve spline array is dimensioned CUR(3,0:2,N) where N is the curve dimension. The second subscript indicates the spline coefficients as follows:

0	r
1	r_{ξ}
2	$r_{\xi\xi}$

The notation is as follows:

CUR (1, ---) : r

CUR (2, ---) : $r_{\xi} = g$

CUR (3, ---) : $r_{\xi\xi} = s$

F : right hand side of Eq.(E-2)

AA : coefficient of s_2 and s_{N-1} for $i=2$ and $N-1$.

BB : coefficient of s_3 and s_{N-2} for $i=2$ and $N-1$.

26. SUBROUTINE DEFAULT (default arrays)

This routine sets the default values in the R,P,SPAN,IMAGE, and TYPE arrays for a block. The routine sweeps the entire block, including the surrounding layer, setting R to "NONE", TYPE to "OUT", and P and SPAN to 0.0. The block, exclusive of the surrounding layer, is then swept again setting IMAGE to 0 and resetting TYPE to "FIELD". The block is identified by NBLK in COMMON/SECTN/, and its size is received in the argument CMAX.

27. SUBROUTINE SETYPE (set type array)

This routine sets the point type array TYPE to the argument IVAL for a section of points in a block identified by START and END in COMMON/SECTN/. This is simply three nested loops over the three curvilinear coordinates, from the minimum of START to END to the maximum thereof, setting TYPE equal to IVAL for all points on the section if IVAL is not "UNFIX" and the point is not currently a "FIX" point. If IVAL is equal to "UNFIX", TYPE is set to "FIELD".

28. SUBROUTINE SETRES (reset restricted Neumann points)

This routine resets the point type array TYPE to the alphanumeric argument IVAL for points currently having TYPE equal to "NEUMANN". The operation is the same as that of SETYPE.

29. SUBROUTINE SETNEU (set Neumann section)

This routine sets up a Neumann boundary section defined by START and END for a block identified by NBLK, these parameters being received in COMMON/SECTN/.

If NEW="YES", a new boundary section is set up to be splined, and therefore the section counter, NEUNUM(NBLK), is incremented for the block indicated. The code stops if this counter exceeds the maximum number of sections allowed, DNEU. (The section here must define a surface in 3D, or a curve in 2D, lying on a boundary.) If ISTART and IEND are omitted from the input statement, the section to be splined defaults to that specified as Neumann boundary points by START and END. Otherwise the splined section is defined by ISTART and IEND and must include

the Neumann section. The three indices of two opposite corners defining the splined section are placed in the array NEUBON(3, 1, NEUNUM(NBLK), NBLK), with $i = 0$ for the first corner and $i = 1$ for the second. These corners are redefined from ISTART and IEND, or from START and END, so that each index for the second corner is not less than the corresponding entry for the first. The curvilinear coordinate that is constant on the section is then determined by examining START and END for equality of corresponding entries, and is placed in NEUNOR (NEUNUM(NBLK), (NBLK)).

The section normal direction, placed in NEUNOR, is made negative if either the section is on an upper side of a block, or it is in the interior and the argument IDIR is not equal to the default "NONE". If SPAVAL is included on the input statement, its value is placed in the array SPAN for each point on the section.

Finally, the classification array, TYPE, is set to "NEUMANN" for each point on the Neumann section defined by START and END that is not already classified as a fixed point, regardless of the value of NEW.

30. SUBROUTINE SETORT (set orthogonal section)

This routine sets up an orthogonal boundary section defined by START and END for a block identified by NBLK, these parameters being received in COMMON/SECTN/.

The section counter, ORTNUM(NBLK), is incremented for the block indicated, and the code stops if the maximum number of sections, DORT, is exceeded. (The section here must define a surface in 3D, or a curve on 2D, lying on a boundary.) The three indices of the two opposite corners of the section, defined by START and END, are placed in the array

ORTBON(3,i,ORTNUM(NBLK),NBLK), with $i = 0$ for the first corner and $i = 1$ for the second. These corners are redefined from START and END, so that each index for the second corner is not less than the corresponding entry for the first. The curvilinear coordinate that is constant on the section is then determined, by examining START and END for equality of corresponding entries, and is placed in ORTNOR(ORTNUM(NBLK),NBLK).

The section normal direction, placed in ORTNOR, is made negative if either the section is on the upper side of a block, or it is in the interior and the argument IDIR is not equal to the default "NONE". The classification array, TYPE, is set to "ORTHOG" for each point on the section not already classified a fixed point. Finally, if SPAVAL is included on the input statement, its value is placed in the array SPAN for each point on the section.

31. SUBROUTINE SETREF (set reflective section)

This routine sets up a reflective boundary section defined by START and END for a block identified by NBLK, these parameters being received in COMMON/SECTN/.

The section counter, REFNUM(NBLK), is incremented for the block indicated, and the code stops if the maximum number of sections, DREF, is exceeded. (The section here must define a surface in 3D, or a curve on 2D, lying on a boundary.) The section is first extended by one point on each side, subject to the limitation of the block boundaries, in each direction for which START and END are not equal by resetting START and END. The three indices of the two opposite corners of the section, defined by START and END, are placed in the array REFBON(3,i,REFNUM

(NBLK),NBLK), with $i = 0$ for the first corner and $i = 1$ for the second. These corners are redefined from START and END, so that each index for the second corner is not less than the corresponding entry for the first. The curvilinear coordinate that is constant on the section is then determined, by examining START and END for equality of corresponding entries, and is placed in REFNUM(REFNUM(NBLK),NBLK). If the section lies on an upper side of the block, the entry in REFNUM is made negative. The classification array, TYPE, is set to "FIELD" for each point on the unextended section not already classified a fixed point. Finally, all points on the surrounding layer that are adjacent to points on the section are classified "REFLECT" in TYPE.

32. SUBROUTINE SETOBJ (set image point)

This routine designates a point as an image point and sets the corresponding object pHere TYPE is set to "IMAGE", for the point identified by IPOINT, and the block number and curvilinear coordinates of the object point identified by NBLK and POINT are put in the array IMAGE for the image point.

33. SUBROUTINE SETSPA (set spacing)

This routine sets values in the spacing array SPAN to the argument SPAVAL on a section of points in a block defined by START and END in COMMON/SECTN/. A set of three nested loops puts the specified off-boundary spacing by SPAVAL in the field array SPAN for all points on the section.

34. SUBROUTINE SETSPL (spline Neumann section)

This routine splines the Neumann boundary sections in a block identified by NBLK in COMMON/SECTN/ and sets up the boundary array RR which contains the points, the two first derivatives, and the cross derivative on the Neumann sections in the block.

If the Neumann section counter, NEUNUM, is non-zero for the block, then the array RR is filled by splining each section indicated for that block. This array is dimensioned $RR(3,0:3,DMRR,SSDB)$, where the second subscript indicates the Cartesian coordinates of the point (0), the first derivatives in the two directions on the section in cyclic order with the curvilinear coordinate that is constant on the section (1,2), and the cross derivative (3). The first subscript indicates the component. The sections in the block are placed in this array point-by-point in a one-dimensional sequence indicated by the third subscript. The value of this subscript, less one, corresponding to the first point on a section is in the index array $IDXRR(DIMB,DNEU)$, in which the first subscript identifies the block and the second subscript identifies the section. A point at (ξ^j, ξ^k) on a section on which ξ^i is constant (with i,j,k cyclic) is then located in the RR array with the third subscript given by the function

$$NDXRR(NBLK,NEU,I1,I2) = IDXRR(NBLK,NEU) + (I2 - 1)*IT1 + I1$$

where B and NEU are the block number and the section number in that block, IT1 is the number of points in the ξ^j direction on the section, and

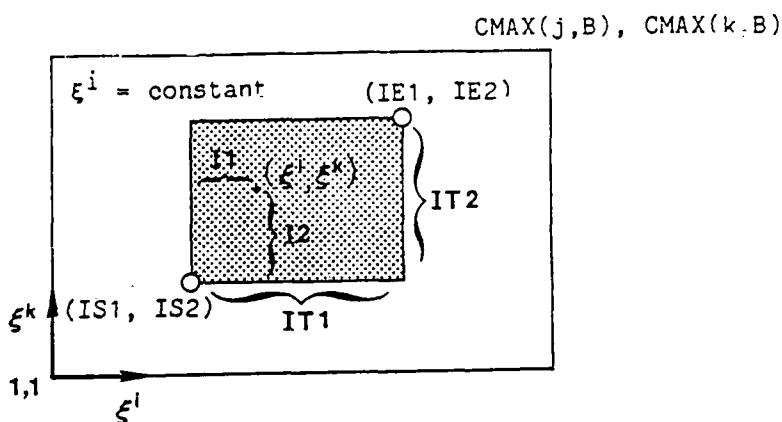
$$I1 = \xi^j - IS2$$

$$I2 = \xi^k - IS2$$

where (IS1,IS2) are the indices of the first corner of the section:

$$IS1 = NEUBON(j,0,NEU,NBLK)$$

$$IS2 = NEUBON(k,0,NEU,NBLK)$$



In a loop over all the Neumann sections that have been created for the block, the curvilinear coordinate that is constant on the section is obtained from NEUNOR, after which NEUNOR is made negative if the section lies on an upper side of the block. The limits of the section, (IS1,IS2) and (IE1,IE2) are obtained from NEUBON, and the number of points on each side are placed in IT1 and IT2.

The section then is swept in two nested loops, placing the three Cartesian coordinates of each point in RR with the second subscript therein as 0, and the third subscript given by index the function NDXRR.

The section is then swept again, placing the Cartesian coordinates of the points on the section in the spline array SUR(3,0:3,DMRR), where the subscripts have the same meaning as the first three in RR, with the

second subscript set at 0 and the third determined from the index function NDXRR with the block number replaced by a 1. The section is then splined by calling SPLSUR, the two first derivatives and the cross derivative being returned in SUR, with the second subscript set at 1 and 2 for the former and at 3 for the latter. A final sweep of the section then transfers these three derivatives from SUR to RR. In the interest of vectorization, these three sweeps of the section are done with the longest side as the innermost loop.

35. SUBROUTINE SETAXS (set right-handed system)

This routine sets the x_3 coordinate on the adjacent layers at $\xi^3 = 0$ and 2 in the third curvilinear direction for a 2D system to ± 1 in such a way as to make the system right-handed (with $x_3 = 0$ or $\xi^3 = 1$) in the block. This is done by setting the x_3 value at $\xi^3 = 2$ to

$$\frac{a_1 \times a_2}{|a_1 \times a_2|}$$

calculated at the first point on $\xi^3=1$ where the denominator is non-zero, and setting x_3 at $\xi^3=0$ to the negative of this value. Here the block is identified by NBLK in COMMON/SECTN/, and its limits are received in the argument CMAX.

36. SUBROUTINE SETR2D (set 2D adjacent planes)

This routine sets the first two Cartesian coordinates, x_1 and x_2 , on the adjacent layers at $\xi^3=0$ and 2 in a block for a 2D system to the current values at $\xi^1=1$. The third coordinate is set to ± 1 in such a way

as to make the system right-handed. Here the block is identified by NBLK in COMMON/SECTN/, and its limits are received in the argument CMAX.

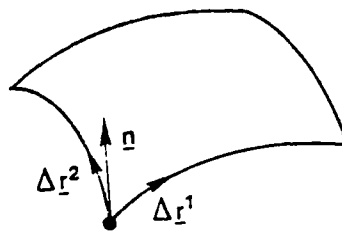
37. SUBROUTINE SETNOR (set reflective normals)

This routine sets the normals to the reflective boundary sections in a block identified by NBLK in COMMON/SECTN/. The routine calculates the unit normal to each section in that block and places it in the array REFDIR(3,DREF,DIMB), where the second subscript gives the section number and the third is the last block number.

The curvilinear coordinate that is constant on the section is recovered as the absolute value of REFNOR(REF,NBLK) where the integer REF is the section number, and the section limits are taken from the array REFBN (3,0,REF,NBLK). The unit normal is then calculated as

$$\underline{n} = \frac{\Delta \underline{r}^1 \times \Delta \underline{r}^2}{|\Delta \underline{r}^1 \times \Delta \underline{r}^2|}$$

where $\Delta \underline{r}^1$ and $\Delta \underline{r}^2$ are the increments from the first corner of the section to the next points in each of the directions on the section:



After being calculated, the normal is given the sign of REFNOR(REF, NBLK), i.e., is made to point into the block.

38. SUBROUTINE SETACC (set uniform acceleration parameters)

This routine sets the values in the acceleration parameter array, ACC, to the argument ACCPAR in a block identified by NBLK in COMMON /SECTN/. The block limits are received in the argument CMAX. The operation is in a set of three nested loops.

39. SUBROUTINE SETIMG

Since only a single block is in core at a time, the image points in a block are grouped according to the object block to minimize the accesses to storage during the setting of values at image points. This subroutine sorts the image points in a block toward that end.

The image points for the block are stacked in the array LINK(-3: 3, DIMI), where the second subscript counts the points. The limit DIMI is set in a parameter statement which can be changed by a global edit. There is a separate array for each block, all of which are kept on storage with the other block arrays. The first subscript identifies the indices of the image point and its object point as follows:

-1,-2,-3 : curvilinear coordinates, ξ^i ($i=1,2,3$), of image point.
0 : object point block number.
1,2,3 : curvilinear coordinates, ξ^i ($i=1,2,3$), of object point.

The array LINI(DIMB) contains the total number of image points in the block, and the array LINB(DIMB) contains the total number of object blocks associated with image points in the block. These three arrays are kept on storage for each block.

The image points for the block, say IB, are sorted in the array LINK so that all points having object points in a single block, say B, are together. The locations of the first and last image points in LINK, i.e., the values of the second subscript thereof, are in the array LINL(2,DIMB,DIMB), where the second subscript is the block number of the block, B, containing the object points and the third subscript is that of the block, IB, containing the image points. The first subscript identifies the first (1) and last (2) locations in LINK of all the image points in block IB having object points in block B.

The subroutine first sweeps the block indicated by the argument IB in three nested loops over the three curvilinear directions, placing all image points, i.e., with TYPE="IMAGE", in the array LINT(-3:3,DIMI) sequentially as they are encountered. The first subscript here functions as does that of the array LINK, and the second simply counts the points as they are entered. The total number of image points in the block is recorded as NT.

A loop is then made over the second subscript of LINT, counting the number of image points having object points in each block, this being recorded in the array LINC(DIMB) for each block. From this information, the starting locations in LINK for each group of image points having object points in a single block are determined and placed in the array

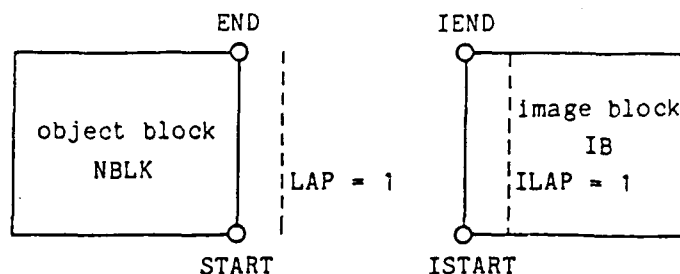
LINP(DIMB). Finally, the image points in the array LINT are sorted into the array LINK by object blocks. The arrays LINK, LINL, LINB, and LINI are returned for later use in setting values at image points.

40. SUBROUTINES SETIMO and SETIMI (image-object correspondence)

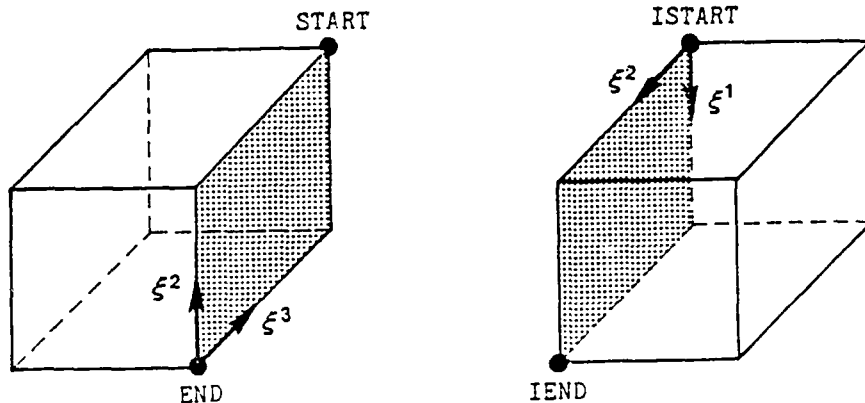
These routines set the image-object correspondence on a cut section in a block (identified by NBLK) defined by START and END in COMMON /SECTN/. The object section of the cut is that in block NBLK defined by START and END, and the image section is in block IB defined by ISTART and IEND, these latter three quantities being received as arguments. All other parameters mentioned below are also received as arguments. (One pair of corresponding entries in START and END, and one pair in ISTART and IEND, will be equal, of course). The object block is treated by SETIMO, and the image block by SETIMI.

In SETIMO, a set of nested loops over the object section defined by START and END sets TYPE equal to "FIELD" for all points on the section for which TYPE is not presently equal to "FIX". If the cut is for an edge in 3D, or a point in 2D, no further action is taken by this subroutine. The section limits for both the object and image sections are saved and then both of these sections are extended by one point off all edges in each direction in which START and END are not equal by re-setting START and END for the object section, and analogously for the image section.

The routine next determines whether the object and image sections are on the 1 or CMAX side of the block in order to locate the appropriate sections on the surrounding layers. The location of the adjacent image surrounding layer outside the object block is set in the array LAP(3), and the object in the image block of this image is set in ILAP:

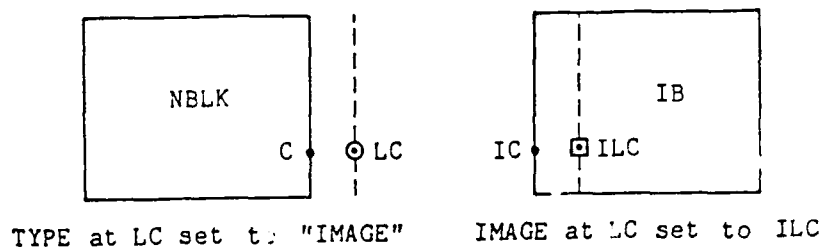


Thus LAP is set to +1 or -1 as the object section is on the CMAX or 1 side of the object block, while ILAP is set to -1 or +1 for the image section on the CMAX or 1 side of the image block. The object section is then swept by a set of three nested loops from the extended START to END. The innermost of those loops is over the curvilinear coordinate direction identified by S(1), etc. Inside this set of loops, there is a progression over the image section from ISTART to IEND with the innermost segment of this progression being in the ξ^1 coordinate direction, etc.:



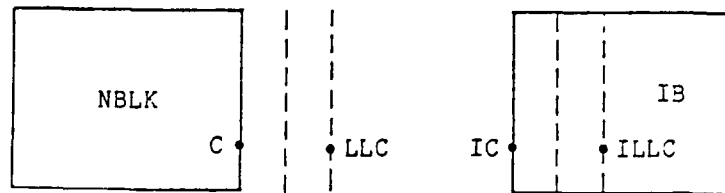
ORDER = 2,3,1

This set of loops sets TYPE equal to "IMAGE" at all points on the adjacent image layer outside the object block and places the curvilinear coordinates of the corresponding object points inside the image block in the IMAGE array:



In these loops, DC(3) and IDC(3) indicate the direction of progression on the object and image sections with values of +1 if END > START, or -1 otherwise, etc. The point on the object section is C(3), and the corresponding point on the image section is IC(3), while the point on the surrounding layer adjacent to C is LC(3) and that adjacent to IC inside

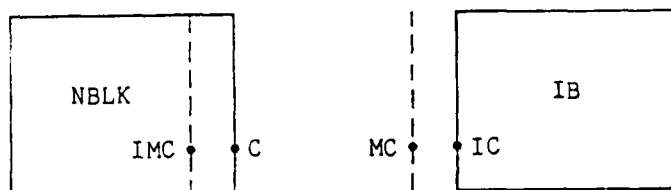
the image block is ILC(3). If a second surrounding layer is called for, this procedure also makes the analogous assignments for this second image layer outside the object block and its corresponding object inside the image block, using LLC(3) and ILLC(3) in the roles of LC and ILC:



Before leaving SETIMO, the section limits on both the object and image sections are restored to their original unextended values.

A similar set of loops in SETIMI sets TYPE equal to "IMAGE" at all points on the image section that have not been classified "FIX". If the cut is for an edge in 3D, or a point in 2D, no further action is taken by this subroutine. If the extent of the cut was contracted by subroutine MOVEIN, the original extent is restored on the image face so that FIX, ORTHOG, and NEUMANN points adjacent to the actual cut can be checked for imaging to points on the object face. The extent of the cut is recontracted before the surrounding layer adjacent to the image face is treated. The section limits are then saved and extended, and TYPE is set to "IMAGE" at all points on the extended adjacent image layer outside the image block. The curvilinear coordinates of the corresponding object points for each of these are placed in the IMAGE array. In these loops, DC and IDC serve as set in SETIMO. Again the point on the object

section is C, and that on the image section is IC. The adjacent point on the layer outside the image block is MC(3), and the corresponding point inside the object block is IMC(3):



TYPE at IC set to "IMAGE"

IMAGE at IC set to C

TYPE at MC set to "IMAGE"

IMAGE at MC set to IMC

Again a second outer layer is set up if called for, using MMC(3) and IMMC(3) in the roles of MC and IMC.

All these operations are done in separate routines for the object block (by SETIMO) and for the image block (by SETIMI) in order to avoid repetitive accesses to the disk.

In SETIMO, TYPE is set to "FIELD" only at all points on the object section defined by START and END that are presently classified as "IMAGE". This prevents points such as "ORTHOG" points, etc., from being reclassified as field points. (Recall that the default for all points on the block is "FIELD".) Also the setting of points on the adjacent

image layer in SETIMO and SETIMI is done only for points presently classified as "OUT". This prevents the reclassification of previously set up image points.

In SETIMI, points on the image section defined by ISTART and IEND are classified "IMAGE" only if the present classification is either "FIELD" or "OUT". This also is to avoid the reclassification of "ORTHO" points, etc.

41. SUBROUTINE NEUIDX (set Neumann index array)

This routine sets up the index array IDXRR for the Neumann boundary points in a block identified by NBLK in COMMON/SECTN/.

In a loop over all the Neumann sections that have been created for the block, the curvilinear coordinate that is constant on the section is obtained from NEUNOR. The limits of the section (IS1,IS2) and (IE1,IE2) are obtained from NEUBON, and the number of points on each side are placed in IT1 and IT2. The current value of the integer DMRR (which was set to 0 before the first section) is entered in the index array IDXRR, DMRR is incremented by the number of points on the section, IT1*IT2, and the total number of Neumann points in all blocks, MXNEU, is incremented by DMRR and is checked against the limit DMNT.

42. SUBROUTINE ORIDX (set orthogonal index array)

This routine sets up the index array IDXRO for the orthogonal boundary points in a block defined by NBLK in COMMON/SECTN/. The operation is the same as that of NEUIDX, with the letters RR replaced by RO and with NEU replaced by ORT in all occurrences.

43. SUBROUTINE JACBCK (check for twisted system)

This routine checks all FIELD points in a block identified by NBLK in COMMON/SECTN/ for zero or negative Jacobians. The limits of the block are received in the argument CMAX. All "FIELD" points in the block are swept in a nest of three loops, and the Jacobian, $\sqrt{g} = a_1 \cdot (a_2 \times a_3)$, is calculated and checked for non-positive values. If any zero values are found, a message is printed to that effect and the code stops unless CHECK has been set to "NO" during the input phase, in which case the initial grid is output and then the code stops. Negative values for the Jacobian result in a message indicating a left-handed system, and the subsequent action is the same as for zero values, except that the code continues if CHECK is not equal to "YES" or "NO".

44. SUBROUTINE EXTCOR (coordinate extrema)

This routine determines the extremes of the Cartesian coordinate values in a block identified by NBLK in COMMON/SECTN/. The block limits are received in the argument CMAX. The extreme values of each of the three Cartesian coordinates are determined in a set of four nested loops. These extremes are printed, and a coordinate scale factor is set to the largest span in the three directions.

45. SUBROUTINE CONFUN (assemble control functions)

This routine calculates the control functions (in the array P) at all FIELD points from the interpolated values of the arc length contribution (in P), the spacing (in SPACE), and the radius of curvature (in RAD) for a block identified by NBLK in COMMON/SECTN/. The block limits

are received in the argument CMAX. This is done in three nested loops, calculating all three control functions in 3D, but only P_1 and P_2 in 2D, as signaled by the argument IDIR.

46. SUBROUTINE EXTCON (control function extrema)

This routine determines the extrema of the control functions in a block identified by NBLK in COMMON/SECTN/. The limits of the block are received in the argument CMAX. The operation is in a set of four nested loops, after which the extrema and their locations are printed.

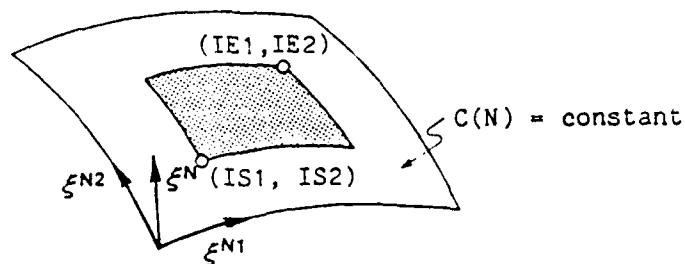
47. SUBROUTINE NEUPTS (values at Neumann points)

This routine sets values of the Cartesian coordinates at Neumann boundary points in a block identified by NBLK in COMMON/SECTN/.

Neumann points are moved on a splined boundary so that the grid is orthogonal to that boundary. The routine checks the Neumann section counter, NEUNUM, for each block for the presence of sections of Neumann points. (The various arrays involved here have been explained in Section II-C5.)

For each Neumann section in block B, the curvilinear coordinate that is constant on the section is obtained as the absolute value of NEUNOR, and the sign index, L, is set to the sign of NEUNOR, this being positive for a lower boundary and negative for an upper. With N set to the curvilinear direction off the surface section, N1 and N2 are set, in cyclic order with N, to the two directions on the section. The value of the (constant) curvilinear coordinate on the section is obtained from

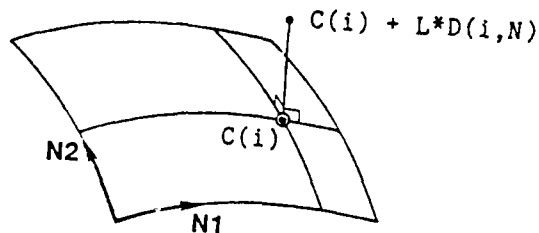
NEUBON with N as the first argument and is placed in $C(N)$. The limits of the section are obtained from NEUBON with $N1$ and $N2$ as the first argument and placed in $(IS1, IS2)$ and $(IE1, IE2)$:



The number of points on each edge of the section is set in $IT1$ and $IT2$ for the two directions on the section.

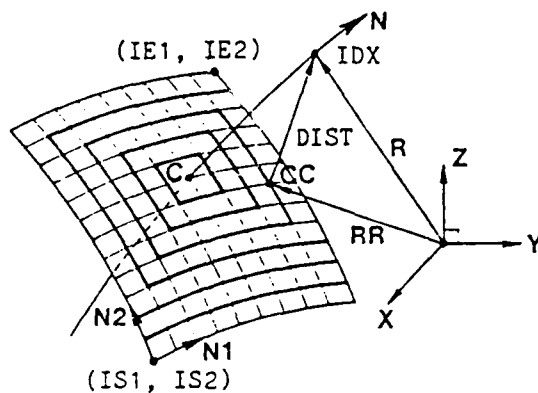
The section is then swept, with the coordinates of the current point on the section set in $C(N1)$ and $C(N2)$. Each point on the section that is designated "NEUMANN" (or with the last N replaced by 1, 2, or 3) is moved on the splined section so that the (straight) grid line from the adjacent point in the field is normal to the section. The foot of this normal is thus the new location of the Neumann point on the section. This is done as follows.

With the curvilinear coordinates of the Neumann point on the section set in $C(i)$ for $i=1, 2, 3$, as described above, those of the adjacent field point are $C(i) + L * D(i, N)$ where $D(i, j)$ is the Kroneker delta array.



These coordinates of the field point are placed in $IDX1, IDX2, IDX3$.

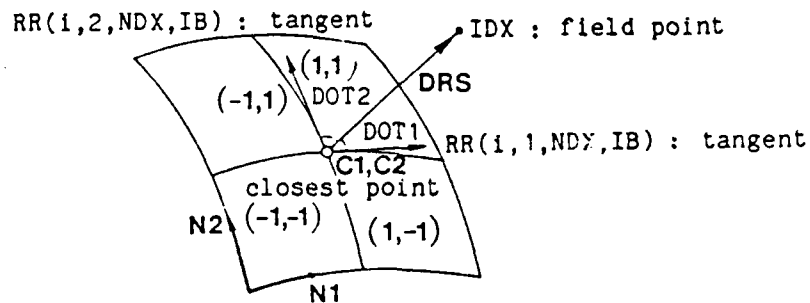
The Neumann point on the section currently closest to this field point is first located and its coordinates are placed in CC . This is done by sweeping the section in ever expanding squares centered on the Neumann point at C . (These squares are actually limited by the section edges, of course, and hence, may become rectangles.)



The Cartesian coordinates of the Neumann point C are obtained from the section array RR with the second subscript 0 and the point location in this array (the third subscript) given by the index function $NDXRR$ in

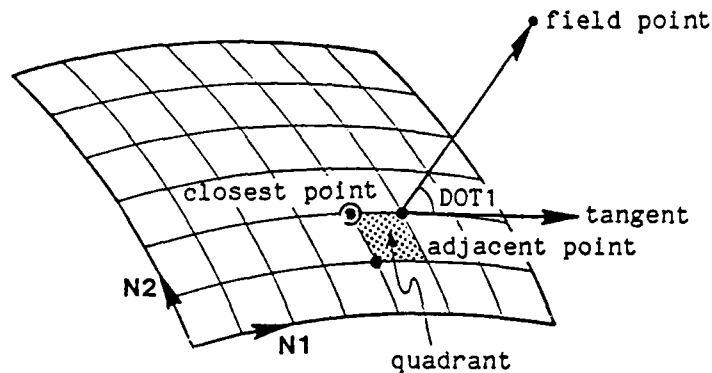
terms of the current relative section coordinates of the point CC, i.e., $CC(N1)-IS1+1$ and $CC(N2)-IS2+1$. The Cartesian coordinates of the field point at $C+L*D$ are obtained from the field array R. The relative section coordinates of the closest point are placed in C1 and C2, and its index in the section array RR is calculated from the index function NDXRR and is placed in NDX.

Next the quadrant about this closest point above which the field point lies is determined by comparing the dot products DOT1 and DOT2 of the vector from the closest point to the field point, DRS (i,N,1), with the tangent vectors to the two grid lines on the section obtained from the section array RR with the second subscript set to 1 and 2. The quadrant is identified by the signs of these two dot products, placed in L1,L2, where values of +1 indicate the upper side in regard to the two directions on the section, and -1 indicates the lower side.



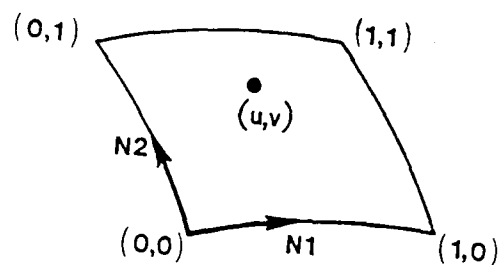
Here L1 or L2 for the third direction in 2D is set to 0.

If either of the points adjacent to the closest points on the quadrant is not a Neumann point, the dot product of the vector from that adjacent point to the field point with the tangent to the grid line on which the adjacent and closest points lie is formed:



If the sign of this dot product indicates that the field point is not over the quadrant (positive sign in the case shown), the appropriate initial guess, X_1 or X_2 , and residual coefficient, CF_1 or CF_2 , is set to zero. (Both of these quantities are discussed below.)

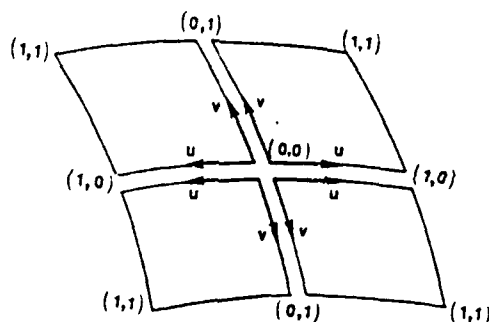
The Neumann boundary point in question is moved to the foot of the normal from the adjacent field point to the surface. This position is found as the solution of the nonlinear system (Eq. F-5) of Appendix F by Newton iteration. The location of the closest current boundary point and the examination of dot products described above has determined the surface cell, i.e., the quadrant, on which this solution lies:



Here u and v vary on the range 0-1 on the quadrant, with values at the corners as shown. (The local coordinate u corresponds to ξ^{N1} and v to ξ^{N2} .) The indices of the four corner points in the section array RB on the quadrant are set as follows from the index function NDXRR:

<u>Corner Index</u>	<u>Coordinates at Corner</u>
NDX00	C1, C2
NDX10	C1+L1, C2
NDX01	C1, C2+L2
NDX11	C1+L1, C2+L2

Since L1 and L2 have been set according to the sign of the dot products described above, this automatically handles all four possible quadrants:



The elements of the matrix Q are set by Eq. (F-1) from the section array RB. Here $\epsilon, \epsilon_\xi, \epsilon_\eta$ and $\epsilon_{\xi\eta}$ are in RB with the second index thereof at 0,1,2, and 3, respectively. These derivatives are multiplied by L1 for ξ and L2 for η to account for the change in the direction of u and v on the different quadrants.

In the iterative solution, the following is the correspondence between the code variables and the notation of Appendix F:

X1,X2	:	u,v (solution)	
G1,G2	:	$F(u), F(v)$	Eq.(F-3)
GP1,GP2	:	$F'(u), F'(v)$	Eq.(F-7)
GPP1,GPP2	:	$F''(u), F''(v)$	Eq.(F-14)
QG2	:	$QF^T(v)$	
QGP2	:	$QF'^T(v)$	
QGPP2	:	$QF''(v)$	
R	:	R	
RP	:	$R - \epsilon = R - F(u)QF^T(v)$	
RU	:	$F'(u)QF^T(v) = \epsilon_u$	
RV	:	$F(u)QF'^T(v) = \epsilon_v$	
RUU	:	$F''(u)QF^T(v) = \epsilon_{uu}$	
RUV	:	$F'(u)QF'^T(v) = \epsilon_{uv}$	
RVV	:	$F(u)QF''(v) = \epsilon_{vv}$	
RUM,RVM	:	$ \epsilon_u , \epsilon_v $	
RURUU,RVRVV	:	$\epsilon_u \cdot \epsilon_{uu}, \epsilon_v \cdot \epsilon_{vv}$	
RURUV,RVRUV	:	$\epsilon_u \cdot \epsilon_{uv}, \epsilon_v \cdot \epsilon_{uv}$	
RMM	:	$\epsilon \cdot \epsilon_v$	
DUU	:	$(R - \epsilon) \cdot [\epsilon_{uu} - (\epsilon_u \cdot \epsilon_{uu})\epsilon_u]$	

$$DVV : (R - r) \cdot [r_{VV} - (r_v \cdot r_{VV})r_v]$$

$$DUV : (R - r) \cdot [r_{UV} - (r_u \cdot r_{UV})r_u]$$

$$DVU : (R - r) \cdot [r_{UV} - (r_v \cdot r_{UV})r_v]$$

$$FU, FV : F1, F2 \text{ (residuals) Eq. (F-8)}$$

$$FUU, FVV, FUV, FVU : \frac{\partial FU}{\partial u}, \frac{\partial FV}{\partial v}, \frac{\partial FU}{\partial v}, \frac{\partial FV}{\partial u} \text{ Eq. (F-12)}$$

$$DET : \frac{1}{\det(J)} \text{ Eq. (F-12)}$$

In order to allow for the 2D case, FU and FV are multiplied by CF1 and CF2, respectively, these by 1 in general but 0 when the corresponding direction is that of invariance in 2D. During the iteration X1 and X2 are confined to the range 0-1, i.e., are not allowed to leave the cell on which the spline coefficients have been set.

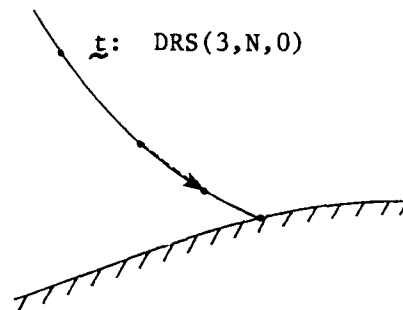
If the Neumann point is classified as "NEUMANi", where the i is 1,2, or 3, the movement of the point on the boundary is restricted to be along lines on which the curvilinear coordinate ξ^i varies if ξ^i is not constant on the surface. This is accomplished by setting the coefficient CF1 or CF2 corresponding to the restricted direction to zero and the corresponding X1 or X2 also to zero.

The iteration continues until the convergence tolerance, FTOL, is reached, or until the maximum allowed number of iteratives, ITM, is reached, in which latter case the code stops. The convergence tolerance FTOL is 1% of the distance from the "FIELD" point (that is adjacent to the Neumann points) to the closest boundary point.

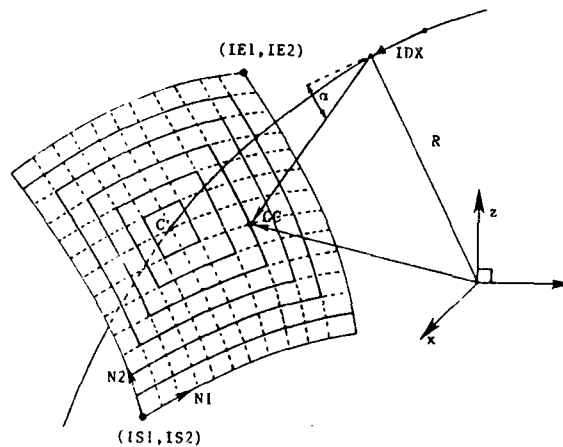
Upon convergence of the iteration, the position of the foot of the normal is calculated from Eq.(F-2), with the notation $V(,1)$ and $V(,2)$ for $F(u)$ and $F(v)$, QB for $QF(v)$, with this new boundary point position being placed in R.

This routine includes extrapolation boundary conditions (zero curvature), as well as the original orthogonal boundary conditions, the former being indicated by the value "EXTRAP" in SPAN.

The unit tangent to the incoming grid line intersecting the surface is calculated from the first two points off the surface and is placed in $DRS(3,N,0)$

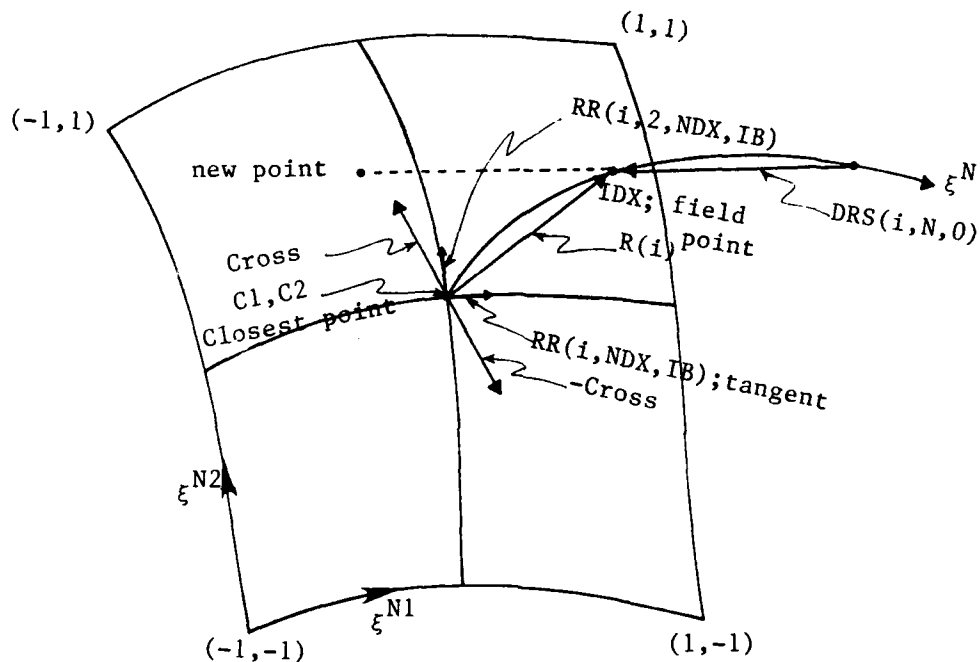


With extrapolation boundary conditions, the "nearest point" search is for the present boundary point making the smallest angle with this unit tangent:



Here $DIST = \sin^2 \alpha$.

The dot products used to determine the quadrant in this case are formed by first forming the cross product of the tangent to the incoming grid line (in $DRS(i,N,0)$) with the vector from the closest current boundary point to the field point (in $DRS(i,N,1)$), and then dotting this vector into the tangent vectors to the two grid lines on the section (in RR with the second subscript as 1 and 2):



Here $DOT1$ is the negative of the dot product of this cross product with the tangent $r_{\xi N2}$ in $DRS(i,N,2)$, and DOT is the dot product with $r_{\xi N1}$ in $DRS(i,N,1)$.

In this case the Newton iteration operates with the system given in Appendix J.

The notation in this case includes some of that given above in this section and the following (cf. Appendix J):

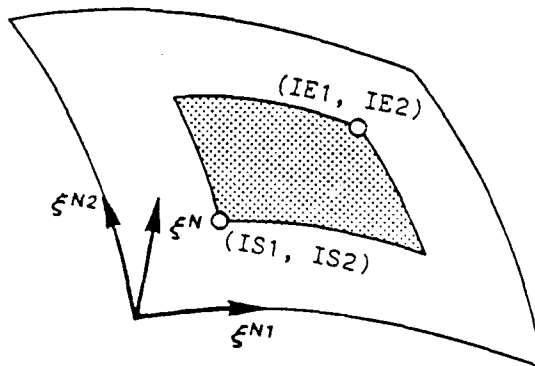
$x_3 : s$
 $F : E \text{ Eq. (J-2)}$
 $DR : J \text{ Eq. (J-4)}$
 $V : F(u), F(v) \text{ Eq. (F3)}$
 $QB : QF^T(v)$

48. SUBROUTINE REFPTS (values at reflective points)

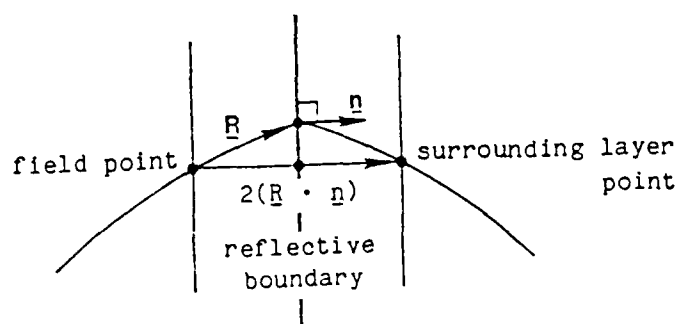
This routine sets values of the Cartesian coordinates at reflective boundary points in a block identified by NBLK in COMMON/SECTN.

The field is swept setting the values at points with TYPE equal to "REFLECT", which lie on the surrounding layer outside the boundary and where the grid is to reflect in the boundary as in a mirror-image.

Each block is examined for reflective point sections, these being numbered in REFNUM. (This and the other arrays involved here have been described in Section II-C8). For each reflective section, the curvilinear coordinate that is constant thereon is obtained as the absolute value of REFNUM, the sign of which indicates a lower or upper boundary as + or -1. The value of the curvilinear coordinate $C(N)$ that is constant on the section is obtained from REFBON, and the section limits are obtained from REFBON:



The section is then swept, determining the normal component of the vector \underline{R} from the interior point adjacent to the boundary point that is adjacent to the point in question, and resetting the reflective point as in the following diagram, where \underline{n} is the unit outward normal:



The notation here is $C(N1)$ and $C(N2)$ for the curvilinear coordinates of the reflective point, $REFDIR$ for the normal \underline{n} , and A for $2(\underline{R} \cdot \underline{n})$.

49. SUBROUTINE R2DPTS (values on adjacent planes in 2D)

This routine sets values of the first two Cartesian coordinates on the adjacent layers in the third direction of a 2D system in a block identified by $NBLK$ in $COMMON/SECTN/$. The last action of this routine is to set the values of the Cartesian coordinates on the layers at $\xi^3=0$ and $\xi^3=2$ equal to those at ξ^1 if the grid is two-dimensional. (The values of the third Cartesian coordinate remain set to -1.0 and $+1.0$, respectively, at $\xi^3=0$ and ξ^3 .)

50. SUBROUTINE AVGCON (control functions at average points)

This routine sets the control functions to zero at AVERAGE points in a block identified by NBLK in COMMON/SECTN/. The field is swept setting the control functions to zero at points with TYPE equal to "AVERAGE".

51. SUBROUTINE REFCON (control functions at reflective points)

This routine sets the control functions at reflective boundary points in a block identified by NBLK in COMMON/SECTN. Each block is examined for sections of reflective points, counted in REFNUM. (This and the other arrays involved here have been discussed in Section II-C8). On each reflective section, the curvilinear direction off the section is obtained as the absolute value of REFNOR, and the section limits are obtained from REFBO. The variable L is set to the sign of REFNOR, i.e., +1 for a lower surface and -1 for an upper. The control functions for the two curvilinear directions on the section are set equal to those at the interior point adjacent to the section and on the same coordinate line. The third function is set to the negative of that at this interior point.

52. SUBROUTINE BONCON (control functions on boundary points)

This routine sets the control functions at fixed, Neumann, and orthogonal boundary points in a block identified by NBLK in COMMON/SECTN by extrapolation from interior values. The control functions at "FIX", "NEUMANN", and "ORTHOG" points are set equal to averages of the values at all the adjacent "FIELD", "IMAGE", and "REFLECT" points (Image

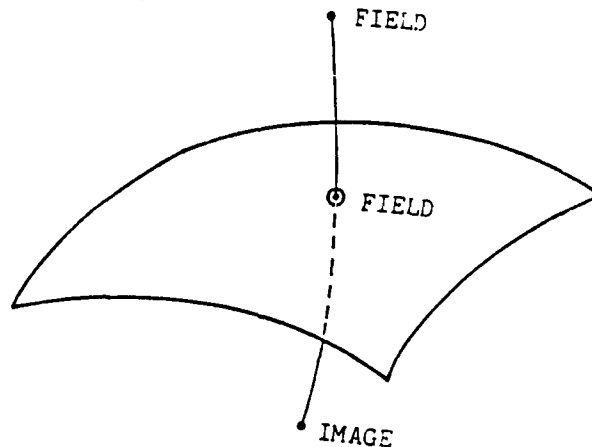
and reflective points on the surrounding outer layers are not included in the average here). Control functions at isolated FIX points are set to zero.

53. SUBROUTINE DFCON (default control functions)

This subroutine simply sets the control functions at all points to "NONE" so that improperly set values can be recognized.

54. SUBROUTINE CUTCON (control functions on cut)

This subroutine sets the control functions on cuts to averages of the values across the cut. The six sides of the block in question are swept, and all three control functions are averaged at all "FIELD" points on the sides. The average includes only curvilinear directions having an "IMAGE" point adjacent to the point on the side, and in thus is across the cut, not along the cut.



55. SUBROUTINE CHKTYP (check for bad field points)

This subroutine checks all "FIELD" points for adjacent "OUT" points and stops if any are found, printing the 26 (8 in 2D) surrounding point classifications.

56. SUBROUTINE CHKSUB (check sub-block coverage)

This subroutine checks a block for a sub-block structure that does not include all points in the block. The routine first initializes the array RAD to 0 for all points in the block, and then sweeps all the sub-blocks in the block resetting RAD to 1. The entire block is then swept again, searching for zero values of RAD at points not classified "OUT".

57. SUBROUTINE CHKINT (check sub-block interior values)

This subroutine checks a block for interior points where values have been set for the Cartesian coordinates, setting ITRANS to "SUB" if any such point is found, or to "ALL" otherwise. The presence of set values means that a sub-block structure is to be assumed for the interpolation for the algebraic grid (cf. Section I-C16).

58. SUBROUTINE CHKSET (check set values)

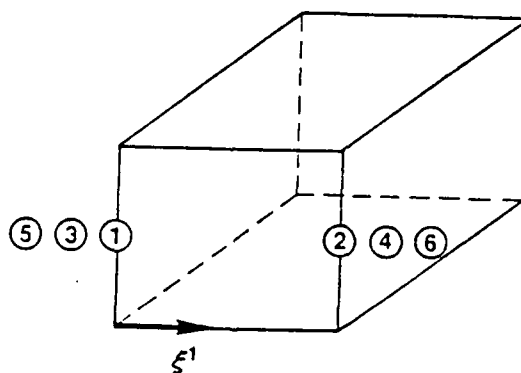
This subroutine checks a block for "FIELD" points that are adjacent to points at which no initial values have been set. This is done by sweeping all points in the block and checking the array R at the 26 adjacent points (8 in 2D) for values that have not been changed from "NONE".

59. SUBROUTINE LIM (image section) - (no longer used)

This routine determines the sections of a block that have image points. The block is identified by the argument IB. The block limits are received as the arguments M1, M2, and M3, and the number of sur-

rounding layers is given by the argument M. A non-zero value of the argument IMGF indicates the presence of image points in the interior of the block. The lower and upper block limits are set in the arrays NLO(3) and NUP(3). For a 2D system, the limits in the third direction are reset to 1.

Since image points are more likely to occur on the block boundaries and on the surrounding layers, the sweep of the block is divided into separate sweeps over each of the boundary sides and the surrounding sides and finally over the interior. There may be two surrounding layers (Section I-B3), and the boundary sides and surrounding sides on which one curvilinear coordinate is constant are numbered as shown below in each direction:



The total number of such sides in each direction is $2(M+1)$, where M is the number of surrounding layers, and this number is placed in the array LNUM(0:3) with the subscript at 1,2, and 3 for the three directions. Also LNUM (0) is set to 1 if there are any image points in the interior of the block, or to 0 otherwise. Thus LNUM gives the number of loops to be done to cover the block. (Some duplication occurs on the edges, of

course.) The limits for each of these loops are placed in the arrays LLO and LUP, which are dimensioned (0:3,3,6). Here the first subscript refers to the direction or the interior (as in LNUM) the last to the side (as in the figure above) and the second subscript refers to the direction to which the limits apply. (For the interior, only a 1 for the last subscript is relevant.)

The sweep of the block then will be conducted as an outer loop of $N=0$ to 3, within which there is a loop of L from 1 to $LNUM(N)$ if $LNUM(N)$ is not zero. Inside this last loop there is a nest of three loops from $LLO(N,i,L)$ to $LUP(N,i,L)$ to cover the particular side (on the interior if $N=0$). In this way all points on each side of the block, and on the surrounding layer, are swept before proceeding to other points, thus minimizing the reading of adjacent blocks into core.

60. SUBROUTINE SSDW (block to file)

This routine writes a block, identified by the integer argument B , onto a disk file. The file number is $BASE+B$, where $BASE$ is an integer set in a `PARAMETER` statement which can be changed by a global edit. The arrays $R,P,TYPE,RAD,SPACE,IMAGE,ACC,RR,RO$, and $LINK$ are written to the file after a rewind.

61. SUBROUTINE SSSR (block from file)

This routine reads a block, identified by the integer argument B, from a disk file. The file number is BASE+B, where BASE is an integer set in a PARAMETER statement which can be changed by a global edit. The arrays R,P,TYPE,RAD,SPACE,IMAGE,ACC,RR,RO, and LINK are read from the file after a rewind.

62. SUBROUTINE SSD1 (coordinate array from file)

This routine reads the coordinate array R for the block identified by the integer argument B from a disk file into the argument array F. The file is the same as used in SSSR and SSDW.

63. SUBROUTINE SSD3 (coordinate, control function, and type array from file)

This routine reads the coordinate array R, the control function array P, and the point type array TYPE for the block identified by the integer argument B from a disk file into the argument arrays F,G, and I, respectively. The file is the same as used in SSSR and SSDW.

64. SUBROUTINE CCDW (block to master core arrays)

This routine writes a block identified by the integer argument B into the master core arrays. The field arrays R, P, SPACE, RAD, IMAGE, ACC, TYPE, RR, RO, and LINK are written into the corresponding storage arrays in COMMON/SSDFL/ as described in Section II-A5.

65. SUBROUTINE CCDD (block from master core arrays)

This routine reads a block identified by the integer argument B from the master core arrays. The field arrays R, P, SPACE, RAD, IMAGE, ACC, TYPE, RR, RO, and LINK are written from the corresponding storage arrays in COMMON/SSDFL/ as described in Section II-A5.

66. SUBROUTINE CCD1 (array from master core array)

This routine reads the argument array F from the argument master core array FF for a block identified by the integer argument B. Here F may correspond to R,P,SPACE, or RAD.

67. SUBROUTINE CCD2 (point type array from master core array)

This routine reads the point type array TYPE from the master core array for the block identified by the integer argument B.

68. SUBROUTINE CCD3 (three arrays from master core array)

This routine reads the argument arrays F, G, and I from the argument master core arrays FF, GG, and II, respectively, for the block identified by the integer argument B. Here F and G may be any of the arrays R,P,SPACE, or RAD, but I must be the TYPE array.

69. SUBROUTINE READF (input points from file)

This routine reads a section of grid points from an input file into the coordinate array R. The Cartesian coordinates of the points in a section of a block identified by START and END in COMMON/SECTN are read from the file by a nest of four loops. The innermost loop is over

N=1,3, reading the Cartesian component RS(N) for nonzero entries in RS. The outer loops are over the three curvilinear coordinates from START to END, with the coordinate S(1) running fastest, etc. Finally, if CLASS="FIX" all the points in the section will be so classified in the array TYPE. The parameters RS and S are received in COMMON/SECTN/, having been equivalenced with RORDER and ORDER in the main program. CLASS is received as an argument. If IPRINT is equal to "YES", the points are printed.

70. SUBROUTINE READL (input points from NAMELIST)

This routine reads a section of grid points from the NAMELIST into the coordinate array R. The operation is the same as that of READF. The Cartesian coordinates appear in the NAMELIST as sequences of values in the one-dimensional array VALUES. The maximum number of points that can be read from the NAMELIST for one section is DIMP, which is set by a PARAMETER statement which can be changed by a global edit. If IPRINT is equal to "YES", the points are printed.

71. SUBROUTINE PRTGRD (print grid)

This routine prints the grid for a section in a block defined by START and END in COMMON/SECTN/. The block is identified by NBLK, also in COMMON/SECTN/. The format of the print is one grid point per line, with the type and the three Cartesian coordinates and control functions printed. The order in which the points are printed is controlled by a permutation of 1,2,3 (not necessarily cyclic) in ORDER, with the curvilinear coordinate indicated by the first entry of ORDER running fastest.

The order in which the Cartesian coordinates appear on the line is controlled by RS in a similar manner, that coordinate indicated by the first entry in RS being listed first. Zero entries in RS caused the omitted coordinate not to be printed. Both S and RS are defaulted to 1,2,3. The parameters RS and S are in COMMON/SECTN/ and are equivalent with RORDER and ORDER, respectively, in the main program.

72. SUBROUTINE WRTPLT (plot file)

This routine writes the Cartesian coordinate array, R, for a block identified by NBLK in COMMON/SECTN/ onto file 8. The block limits are received in the argument CMAX. The operation is in a set of three nested loops, with an implied loop for the write. The form is unformatted if FORM is "UNFORM", E20.8 format for "E", or list-directed "LIST".

73. SUBROUTINE WRTXYZ (file grid in three arrays)

This routine writes the three Cartesian coordinates unformatted onto file 9 in three separate arrays for a block identified by NBLK in COMMON/SECTN/. The block limits are received in the argument CMAX. The operation is in three nested loops setting values in the three arrays X,Y, and Z from the coordinate array R, after which these three arrays are written to the file.

74. SUBROUTINE WRTGRD (file grid with image arrays)

This routine writes the point type array TYPE, the image array IMAGE, and the coordinate array R, unformatted onto a file identified by the integer argument GRIDFIL for a block identified by NBLK in COM-

MON/SECTN/. The block limits are received in the argument CMAX. The operation is in three nested loops, with implied loops over the components. All the surrounding layers are included.

75. SUBROUTINE WRTRRR (file grid only)

This routine operates as does WRTGRD, except that only the coordinate array, R, is written.

76. SUBROUTINE WRTRES (with restart file)

This routine writes the field arrays R, P, TYPE, RAD, SPACE, IMAGE, ACC, RR, RO, and LINK on file 7 for restart.

77. SUBROUTINE REDRES (reads restart file)

This routine reads the field arrays R, P, TYPE, RAD SPACE, IMAGE, ACC, RR, RO, and LINK from file 7 for restart.

78. SUBROUTINE AITKEN

This subroutine receives the current and preceding values of an iterate and the residual in the solution of a nonlinear equation and returns a new iterate for Newton-Raphson iteration. The current iterate and residual are received as the real arguments X and F, with the preceding values in the real arguments XO and FO. The new iterate, returned in X, is calculated from

$$X = X - AC \frac{X - XO}{F - FO} F$$

where a parameter (range 0-1) is received as the real argument AC. (At the first call, X is simply returned as 1.01X.) The quotient in parentheses is bounded by unity and is also set to unit when the residual falls below a set minimum ($FMIN = 10^{-8}$). Each iterate will be printed if the integer argument IPRT is nonzero.

79. SUBROUTINE SING, COSG, TANG, ACOSG

These functions return the hyperbolic sine, cosine, tangent, and inverse cosine if the real parameter FLAG, transmitted through COMMON/HYPER/, is less than unit for the first three, and if the magnitude of the argument is greater than unity for the last. Otherwise the circular functions are returned. The angle is received as the real argument ANG in radians for the first three and as the real value A for the last.

80. SUBROUTINE CHKNEU (checks for bad Neumann points)

This subroutine checks all "NEUMANN" points for adjacent "OUT" points and stops if any are found, printing the 26 (8 in 2D) surrounding point classifications.

81. SUBROUTINE CHKORT (checks for bad orthogonal points)

This subroutine checks all "ORTHO" points for adjacent "OUT" points and stops if any are found, printing the 26 (8 in 2D) surrounding point classifications.

82. SUBROUTINE CHKREF (checks for bad reflective points)

This subroutine checks all "REFLECT" points for adjacent "OUT" points and stops if any are found, printing the 26 (8 in 2D) surrounding point classifications.

83. SUBROUTINE CHKCON (checks set control function values)

This subroutine checks a block for "FIELD" points that are adjacent to points at which no control functions have been set. This is done by sweeping all points in the block and checking the array P at the 6 directing adjacent points (4 in 2D) for values that have not been changed from "NONE".

84. SUBROUTINE READB (reads an entire file)

This routine reads a boundary segment from an input file into the storage array BDATA(3,DIMD) when an entire file of segments is being read at once. The total number of points on the entire file is received as the argument NTOT. The reading is done as in READF (Section II-F69).

85. SUBROUTINE READS (input points from storage array)

This routine transfers a section of grid points from the storage array BDATA into the coordinate array R. The operation is as in READF (Section II-F69), with the section identified by BLOCK, START, and END. The location in BDATA of the first point on the segment is received as the argument LSEG. The points on the section are classified also as in READF.

86. SUBROUTINE BLKCON (Checks for possible loss of continuity)

This subroutine checks each block for image points having adjacent image points that have non-field points in another block as object points.

The routine uses the linkage arrays in COMMON/ILINK/ as does IMCPTS (Section II-F5). All of the image points are checked for adjacent image points. If the object block for such an adjacent image point is different from the block being checked, the TYPE array for that object block is read into NTYPE by calling SSD3 or CCD2 as appropriate. If the object point is not a field point, a warning is printed.

87. SUBROUTINE MOVEIN (Contracts cut section)

This routine contracts a cut section if appropriate. The curvilinear coordinate that is constant on the section is identified as ξ^N , and the two coordinates that vary on the surface are ξ^{N1} and ξ^{N2} , where N,N1,N2 are cyclic. If no "FIELD" point, or an "IMAGE" point not adjacent to an "OUT" point outside the block, is found on an edge of the section, that edge is moved in one point.

88. SUBROUTINE SURSYS (surface elliptic grid generation system)

This subroutine operates as does VOLSYS, but with the surface elliptic grid generation system discussed in Appendix K, in place of the system given in VOLSYS. This surface system is given by Eq. (K-31) and the concomitant equations in Appendix K. Since the grid is being generated on a surface, the third curvilinear coordinate, C3, is set to 1, and the sweep is over C1 and C2.

The routine has the two surface parametric coordinates, u and v , in the array R , with the first subscript 1 and 2, instead of the Cartesian coordinates. The derivatives in DR are thus derivatives of the parametric coordinates:

$$u_{\xi^i} = DR(1,i,0)$$

$$v_{\xi^i} = DR(2,i,0)$$

$$u_{\xi^i \xi^i} - 2u = DR(1,i,1)$$

$$v_{\xi^i \xi^i} - 2v = DR(2,i,1)$$

$$u_{\xi^i \xi^j} = DR(1,i,j)$$

$$v_{\xi^i \xi^j} = DR(2,i,j)$$

where i and j assume either of the values 1 and 2, corresponding to the curvilinear coordinates, ξ^α and ξ^β (Appendix K), that vary on the surface.

The derivatives of the Cartesian coordinates with respect to the parametric coordinates are interpolated from the spline by calling `SPLINT`, these being returned in the array `VR`:

$$r_u = : VR(_,1,0)$$

$$r_v = : VR(_,2,0)$$

$$r_{uu} = : VR(_,1,1)$$

$$\Gamma_{vv} = : VR(_,2,2)$$

$$\Gamma_{uv} = : VR(_,1,2)$$

The form of the equations is the same as that for the 2D form of those used in VOLSYS with the following identifications:

$$GG(1,1) = AA = a$$

$$GG(2,2) = CC = c$$

$$GG(1,2) = -BB = -b$$

and with the addition of a non-zero right-hand side:

$$SUM3 : J_v^2 \Delta_2 u \text{ or } J_v^2 \Delta_2 v$$

The code notation for the remaining quantities in the system is as follows (cf. Eq. (K-28)-(K-34)):

$$a : AA$$

$$b : BB$$

$$c : CC$$

$$J_v^2 : SJNU$$

$$\bar{g}_{\alpha\alpha} : GAA$$

$$\bar{g}_{\beta\beta} : GBB$$

$$\bar{g}_{\alpha\beta} : GAB$$

$$\bar{J}_v : JBN$$

$$\Delta_2^u, \Delta_2^v : BEL(1), BEL(2)$$

$$(\bar{g}_{\alpha\alpha})_u, (\bar{g}_{\alpha\alpha})_v : GAAU, GAAV$$

$$(\bar{g}_{\beta\beta})_u, (\bar{g}_{\beta\beta})_v : \text{GBBU, GBBV}$$

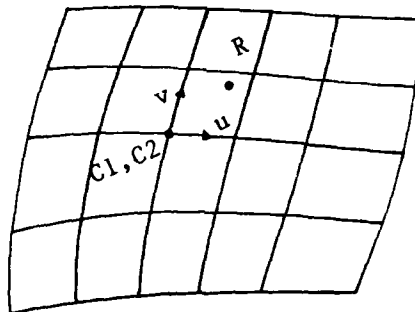
$$(\bar{g}_{\alpha\beta})_u, (\bar{g}_{\alpha\beta})_v : \text{GABU, GABV}$$

$$(\bar{J}_v)_u, (\bar{J}_v)_v : \text{JBNU, JBNV}$$

89. SUBROUTINE SPLINT (Interpolation on surface spline)

This subroutine receives values of the two surface parametric coordinates in the argument array R (with the first subscript 1 and 2) at the grid point with argument indices I1, I2, and interpolates for the derivatives, r_u , etc., returning then in the array VR through COMMON/SURFACE/. The spline is received in the array RE through COMMON/SURFACE/.

With reference to the following figure,



since the spline is with reference to the indices of the surface as read in, i.e., the point counters in each direction (ranging from 1 to the surface dimensions), the particular section of the spline on which the point is located is determined by $C1 = \text{IFIX}(R(1, \dots))$, $C2 = \text{IFIX}(R(2, \dots))$. The spline coordinates (u and v , range 0-1) on this section are then determined by subtracting these integer values from the values received in R.

The spline is set up as discussed in Appendix F, so that, from Eq. (F-2),

$$\underline{r}(u,v) = F(u)QF^T(v)$$

The derivatives then are

$$\underline{r}_u = F'QF^T : VR(_,1,0)$$

$$\underline{r}_v = FQF^{T'} : VR(_,2,0)$$

$$\underline{r}_{uu} = F''QF^T : VR(_,1,1)$$

$$\underline{r}_{vv} = FQF^{T''} : VR(_,2,2)$$

$$\underline{r}_{uv} = F'QF^{T'} : VR(_,1,2)$$

Since the generation system is homogeneous in u and v , it is not necessary to reference these derivatives to the full range of the parametric coordinates; rather the 0-1 range of u and v on the spline section is sufficient.

The following notation is used in the code:

u,v	:	$X1,X2$
Q	:	Q
$F(u),F^T(v)$:	$G1,G2$
$F',F^{T'}$:	$G1P,G2P$
$F'',F^{T''}$:	$G1PP,G2PP$
$QF^{T'}$:	$QGP2$
$QF^{T''}$:	$QGPP2$

90. SUBROUTINE SURCOR (Cartesian coordinates from spline)

This subroutine receives the two surface parametric coordinates of all grid points on a surface grid in the argument array R (with first subscript 1 and 2) and returns the three Cartesian coordinates of the grid points in the same array (but with the first subscript 1,2,3).

For each point on the grid the routine calls SPLINT to interpolate on the spline for the Cartesian coordinates of the point, placing these values in the array RAD. After the entire grid has been swept, all the values in RAD are transferred to R for return.

91. SUBROUTINES MERP2, MERP3

These subroutines are identical to TERP2 and TERP3, except that the result has the opposite sign.

92. SUBROUTINE DERP3

This subroutine is identical to TERP3, except that the result has the opposite sign and is multiplied by 2.0.

APPENDIX A

TRANSFINITE INTERPOLATION

In two dimensions, the Lagrange interpolation functions written individually in each curvilinear direction are

$$r(\xi, \eta) = \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) r(\xi_n, \eta) \quad \left(\begin{array}{c} \xi \\ \hline \end{array} \right) \quad (1a)$$

and

$$r(\xi, \eta) = \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) r(\xi, \eta_m) \quad \left(\begin{array}{c} \eta \\ \hline \end{array} \right) \quad (1b)$$

where the "blending functions", ϕ_n and ψ_m , satisfy the conditions

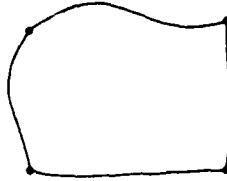
$$\begin{aligned} \phi_1(0) &= 1, & \phi_2(0) &= 0 \\ \phi_1(1) &= 0, & \phi_2(1) &= 1 \\ \psi_1(0) &= 1, & \psi_2(0) &= 0 \\ \psi_1(1) &= 0, & \psi_2(1) &= 1 \end{aligned}$$

(In this appendix, ξ, η , and ζ are defined on the ranges 0-I, 0-J, and 0-K, respectively.) This interpolation is called "transfinite" since it matches the entire boundary defined by $\xi=0$ and $\xi=I$ in the first equation, or by $\eta=0$ and $\eta=J$ in the second, i.e., at a nondenumerable number of points (cf. Ref. 13 and 14).

The tensor product form

$$\underline{r}(\xi, \eta) = \sum_{n=1}^2 \sum_{m=1}^2 \phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi_n, \eta_m) \quad (2)$$

matches the four corners:



It does not, however, match all the boundary.

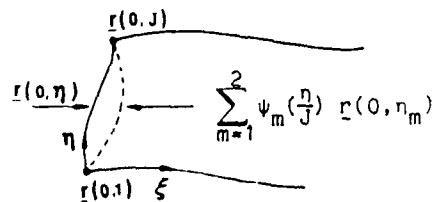
The sum of Eq. (1a) and (1b),

$$\underline{S}(\xi, \eta) = \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) \underline{r}(\xi_n, \eta) + \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi, \eta_m) \quad (3)$$

when evaluated on the $\xi=0$ boundary gives

$$\underline{S}(0, \eta) = \underline{r}(0, \eta) + \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) \underline{r}(0, \eta_m)$$

This does not match the $\xi=0$ boundary because of the second term on the right, which is an interpolation between the ends of this boundary:



Similar effects occur on all the other boundaries, and the discrepancy on the $\xi=1$ boundary is

$$\sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) \underline{r}(1, \eta_m)$$

The discrepancies on both of these boundaries can be removed by subtracting from $\underline{S}(\xi, \eta)$ a function formed by interpolating the discrepancies between the two boundaries:

$$\underline{R}(\xi, \eta) = \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) \left[\sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi_n, \eta_m) \right] \quad (4)$$

But this is simply the tensor product form given by Eq. (2), which matches the four corners.

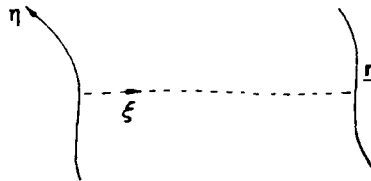
The function $\underline{S}-\underline{R}$ then matches all four sides of the boundary, so that we have the transfinite interpolation from,

$$\begin{aligned} \underline{r}(\xi, \eta) &= \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) \underline{r}(\xi_n, \eta) + \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi, \eta_m) \\ &- \sum_{n=1}^2 \sum_{m=1}^2 \phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi_n, \eta_m) \end{aligned} \quad (5)$$

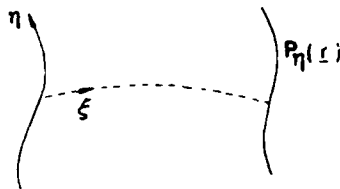
which matches the entire boundary. By contrast, the tensor product form, Eq. (2), matches only the four corners on the boundary.

Projectors

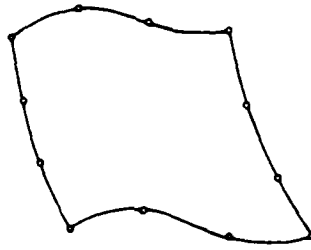
Now let $P_{\xi}(\underline{r})$ be a one-dimensional interpolation function ("projector") in the ξ -direction which matches \underline{r} on the two lines $\xi = \xi_n$ ($n=1,2$):



(Note that the subscript ξ here does not denote differentiation.) Similarly, let $P_{\eta}(\underline{r})$ match \underline{r} on the two lines $\eta = \eta_m$ ($m = 1,2$). Projectors are discussed in more detail in Ref. (13 and 14). Some discussion is also given in Ref. (1). The product projector, $P_{\xi}[P_{\eta}(\underline{r})]$, then matches the function $P_{\eta}(\underline{r})$, instead of \underline{r} , on the two lines $\xi = \xi_n$:



Then, since $P_{\eta}(\underline{r})$ matches \underline{r} on the two lines $\eta = \eta_m$, it follows that the product projector will match \underline{r} at the four corner points (ξ_n, η_m) :



Clearly the same conclusion is reached for the product projector $P_\eta[P_\xi(\underline{r})]$, so that the projectors P_ξ and P_η commute.

The sum projector, $P_\xi(\underline{r}) + P_\eta(\underline{r})$, matches $\underline{r} + P_\eta(\underline{r})$ on the two lines $\xi=\xi_n$, and matches $\underline{r} + P_\xi(\underline{r})$ on the two lines $\eta=\eta_m$. It should be clear then that the projector, $P_\xi(\underline{r}) + P_\eta(\underline{r}) - P_\xi[P_\eta(\underline{r})]$, will match \underline{r} on the two lines $\xi=\xi_n$, since $P_\xi[P_\eta(\underline{r})]$ matches $P_\eta(\underline{r})$ on these lines. Similarly, the projector $P_\xi(\underline{r}) + P_\eta(\underline{r}) - P_\eta[P_\xi(\underline{r})]$ matches \underline{r} on the two lines $\eta=\eta_m$. Therefore, since $P_\xi P_\eta = P_\eta P_\xi$, the Boolean sum projector,

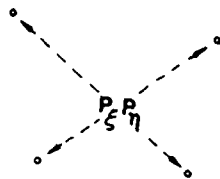
$$P_\xi \oplus P_\eta = P_\xi + P_\eta - P_\xi P_\eta$$

will match \underline{r} on the entirety of the four intersecting lines, $\xi=\xi_n$ and $\eta=\eta_m$, which is, of course, the entire boundary.

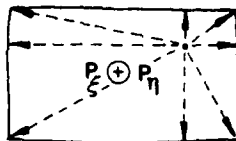
In summary, the individual projectors, P_ξ and P_η , interpolate unidirectionally between two opposing boundaries:



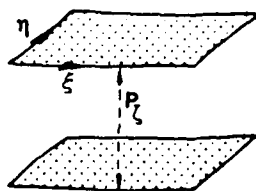
The product projector, $P_\xi P_\eta$, interpolates in two directions from the four corners:



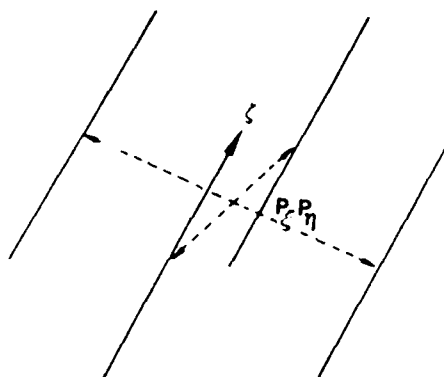
The Boolean sum projector, $P_{\xi} \oplus P_{\eta}$, interpolates from the entire boundary:



In three dimensions, the individual projectors, P_{ξ} , P_{η} , and P_{ζ} , interpolate unidirectionally between two opposing faces of the six-sided region:



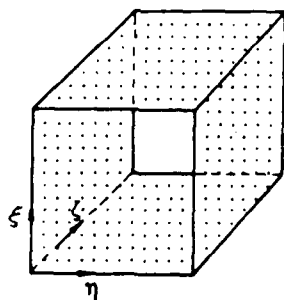
(matching \underline{r} on each of the two faces in each case). The double product projector, $P_{\xi} P_{\eta}$, interpolates in two directions from the four edges along which ξ and η are constant:



(matching \underline{r} on each of these edges). The Boolean sum projector

$$P_{\xi} \oplus P_{\eta} = P_{\xi} + P_{\eta} - P_{\xi} P_{\eta} \quad (6)$$

interpolates in two directions from the four faces on which either ξ or η is constant:

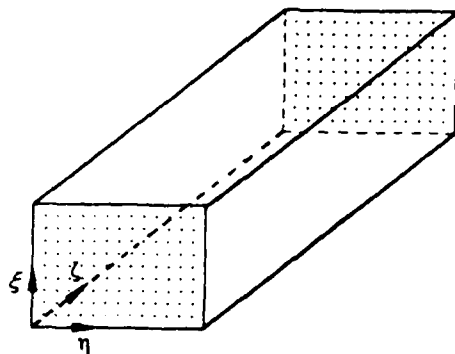


(matching \underline{r} on all of these faces).

The Boolean sum projector

$$P_{\xi} P_{\eta} \oplus P_{\zeta} = P_{\xi} P_{\eta} + P_{\zeta} - P_{\xi} P_{\eta} P_{\zeta} \quad (7)$$

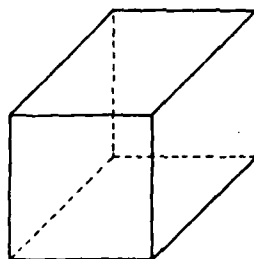
interpolates in three directions, matching \underline{r} on the four edges on which ξ and η are constant and also on the two faces on which ζ is constant:



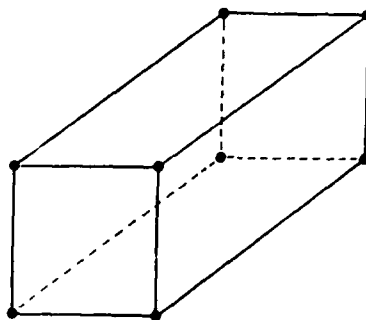
The Boolean sum projector

$$P_{\xi}P_{\eta} \oplus P_{\eta}P_{\zeta} \oplus P_{\zeta}P_{\xi} = P_{\xi}P_{\eta} + P_{\eta}P_{\zeta} + P_{\zeta}P_{\xi} - 2P_{\xi}P_{\eta}P_{\zeta} \quad (8)$$

interpolates in three directions, with \underline{r} matched only on all 12 edges:



The triple product projector, $P_{\xi}P_{\eta}P_{\zeta}$, interpolates \underline{r} from the eight corners:



Finally the Boolean sum projector

$$P_{\xi} \oplus P_{\eta} \oplus P_{\zeta} = P_{\xi} + P_{\eta} + P_{\zeta} - P_{\xi}P_{\eta} - P_{\eta}P_{\zeta} - P_{\zeta}P_{\xi} + P_{\xi}P_{\eta}P_{\zeta} \quad (9)$$

matches r on the entire boundary.

Much cancellation occurs in the algebraic manipulation of the projectors involved in developing the above relations, since $P_{\xi}P_{\xi} = P_{\xi}$, etc. Thus, for example,

$$P_{\xi} \oplus P_{\xi}P_{\eta} = P_{\xi} + P_{\xi}P_{\eta} - P_{\xi}P_{\xi}P_{\eta} = P_{\xi} + P_{\xi}P_{\eta} - P_{\xi}P_{\eta} = P_{\xi}$$

This is to be expected since interpolation by P_{ξ} matches the function on all of the two sides on which ξ is constant, while $P_{\xi}P_{\eta}$ matches the function on the four edges on which ξ and η are constant. But these edges are contained on the two sides cited, so that nothing is changed by adding $P_{\xi}P_{\eta}$ to P_{ξ} in the Boolean sense.

The importance of the projectors is that the structure given above allows multi-directional interpolation to be constructed systematically from unidirectional forms. With one-dimensional interpolation of the form of Eq. (1) we have

$$P_{\xi}(\underline{r}) = \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) \underline{r}(\xi_n, \eta) \quad (10a)$$

$$P_{\eta}(\underline{r}) = \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi, \eta_m) \quad (10b)$$

so that

$$\begin{aligned} P_{\xi} P_{\eta}(\underline{r}) &= P_{\xi}[P_{\eta}(\underline{r})] \\ &= \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) \left[\sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi_n, \eta_m) \right] \\ &= \sum_{n=1}^2 \sum_{m=1}^2 \phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) \underline{r}(\xi_n, \eta_m) \end{aligned} \quad (11)$$

which is just the tensor product form given previously in Eq. (2), so that the two-directional transfinite interpolation corresponding to the projector $P_{\xi} \oplus P_{\eta}$ is just that given by Eq. (5).

The triple product corresponding to Eq. (10) is simply

$$P_{\xi} P_{\eta} P_{\zeta}(\underline{r}) = \sum_{n=1}^2 \sum_{m=1}^2 \sum_{\ell=1}^2 \phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) \theta_{\ell}\left(\frac{\zeta}{K}\right) \underline{r}(\xi_n, \eta_m, \zeta_{\ell}) \quad (12)$$

Although Hermite interpolation can be defined in terms of additional points, and thus be put in this same form also, the use of projectors allows a more direct statement as follows. For the projectors we have,

$$P_{\xi}(r) = \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) r(\xi_n, \eta) + \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) r_{\xi}(\xi_n, \eta) \quad (13a)$$

and

$$P_{\eta}(r) = \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) r(\xi, \eta_m) + \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) r_{\eta}(\xi, \eta_m) \quad (13b)$$

where the blending functions satisfy

$$\phi_1(0) = 1, \quad \phi_2(0) = 0$$

$$\phi_1(1) = 0, \quad \phi_2(1) = 1$$

$$\phi_1'(0) = \phi_1'(1) = \phi_2'(0) = \phi_2'(1) = 0$$

$$\phi_1(0) = \phi_1(1) = \phi_2(0) = \phi_2(1) = 0$$

$$\phi_1'(0) = 1, \quad \phi_2'(0) = 0$$

$$\phi_1'(1) = 0, \quad \phi_2'(1) = 1$$

with analogous conditions for ψ and Ψ . Now

$$\begin{aligned}
P_{\xi} P_{\eta}(r) &= P_{\xi}[P_{\eta}(r)] \\
&= \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) \left[\sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) r(\xi_n, \eta_m) + \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) r_{\eta}(\xi_n, \eta_m) \right] \\
&+ \sum_{n=1}^2 \phi_n\left(\frac{\xi}{I}\right) \left[\sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) r_{\xi}(\xi_n, \eta_m) + \sum_{m=1}^2 \psi_m\left(\frac{\eta}{J}\right) r_{\xi\eta}(\xi_n, \eta_m) \right] \\
&= \sum_{n=1}^2 \sum_{m=1}^2 \left[\phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) r(\xi_n, \eta_m) \right. \\
&+ \phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) r_{\eta}(\xi_n, \eta_m) + \phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) r_{\xi}(\xi_n, \eta_m) \\
&+ \left. \phi_n\left(\frac{\xi}{I}\right) \psi_m\left(\frac{\eta}{J}\right) r_{\xi\eta}(\xi_n, \eta_m) \right] \tag{14}
\end{aligned}$$

Then the two-directional transfinite interpolation can be constructed by substitution of Eq. (13) and (14) into the projector $P_{\xi} \oplus P_{\eta}$. Here the tensor product form $P_{\xi} P_{\eta}$ interpolates from the values of the function, its two first derivatives, and the cross-derivative at the four corners of the boundary. The transfinite interpolation form, $P_{\xi} \oplus P_{\eta}$, however, interpolates from the value of the function and its normal derivative on the entire boundary.

The above evaluations of the product projectors serve to illus-

trate the evaluation of such products for general projectors, i.e., that the effect of the product projectors is simply an interpolation in one-direction of an interpolant in another direction. This allows the multi-directional transfinite interpolation to be constructed from the Boolean sums of the projectors given above using any appropriate unidirectional interpolation forms as the basis projectors. It should also be noted that the unidirectional interpolation does not have to be of the same form in all the directions. Thus Lagrange interpolation could be used in one direction while Hermite is used in the other direction of a two-dimensional construction. Also, the blending functions do not have to be polynomials. This freedom to combine different types of univariate interpolation gives considerable flexibility to transfinite interpolation based on the projector structure.

Coding

To generalize the notation for coding, let the blending functions ϕ_n and ϕ_n in Eq. (13a) be given by $B_{0n}^{(1)}$ and $B_{1n}^{(1)}$, respectively, and let ψ_n and ψ_n of Eq. (13b) be given by $B_{0n}^{(2)}$ and $B_{1n}^{(2)}$, etc. The four blending functions in the direction 1 can then be represented as $B_{mn}^{(1)}$ where the first subscript indicates the function associated with r ($m=0$) or $r_{\xi 1}$ ($m=1$), while the second subscript indicates the end of the interpolation line ($n=0$ for the first end and $n=1$ for the other). All 12 blending functions are then represented by

$$B_{mn}^{(1)}(\xi) \quad \begin{aligned} i &= 1, 2, 3 \\ m &= 0, 1 \\ n &= 0, 1 \end{aligned}$$

Also let \underline{r} and its derivatives be represented as $\underline{R}^{(\underline{D})}(\underline{C})$ where the superscript is a 3-vector indicating the derivative, e.g.,

$$\underline{D} = 0,0,0 \rightarrow \underline{r}$$

$$\underline{D} = 1,0,0 \rightarrow \underline{r}_{\xi^1}$$

$$\underline{D} = 0,1,0 \rightarrow \underline{r}_{\xi^2}$$

$$\underline{D} = 1,1,0 \rightarrow \underline{r}_{\xi^1 \xi^2}$$

$$\underline{D} = 1,1,1 \rightarrow \underline{r}_{\xi^1 \xi^2 \xi^3}, \text{ etc.}$$

and the argument is a 3-vector indicating the point of evaluation, i.e., $\underline{\xi}$ with one or more entries replaced by an end value.

The one-dimensional projector in the i -direction for Hermite interpolation can then be written

$$P^i = \sum_{m=0}^1 \sum_{n=0}^1 B_{mn}^{(i)}(\underline{\xi}) \underline{R}^{(\underline{D}_m)}(\underline{C}_n) \quad (15)$$

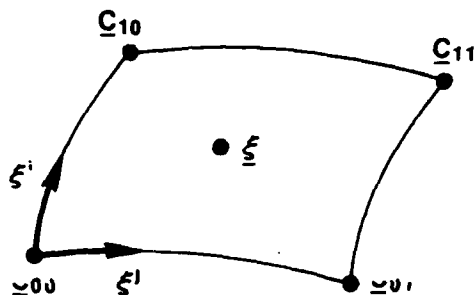
where the first summation is over the function forms and the second is over the ends. Here $\underline{D}_0 = 0,0,0 \rightarrow \underline{r}$, while the i -entry of \underline{D}_1 is 1 and the other two entries are zero, indicating the derivative \underline{r}_{ξ^i} . Also, \underline{C}_0 and \underline{C}_1 , have their i -entries equal to $\text{START}(i)$ and $\text{END}(i)$, respectively, while the other two entries are equal to the corresponding entries of $\underline{\xi}$:



The double product projector then is

$$p^i p^j = \sum_{m=0}^1 \sum_{n=0}^1 \sum_{p=0}^1 \sum_{q=0}^1 B_{mn}^{(i)}(\xi) B_{pq}^{(j)}(\xi) R_{mp}^{(D)}(C_{nq}) \quad (16)$$

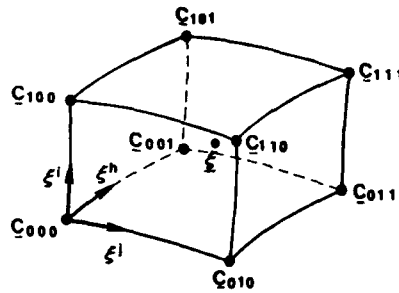
where $D_{00} = 0, 0, 0 \rightarrow r$. Now the i -entry of D_{10} is 1, with the others zero, indicating r_{ξ^i} , while the j -entry of D_{01} is 1, with the others zero, indicating r_{ξ^j} . Also both the i and j entries of D_{11} are 1, and the other entry is zero, indicating the cross derivative, $r_{\xi^i \xi^j}$. The i -entries of C_{0q} and C_{1q} are equal to $START(i)$ and $END(i)$, respectively, while the j -entries of C_{n0} and C_{n1} are equal to $START(j)$ and $END(j)$:



Similarly, the triple product projector is

$$p^i p^j p^k = \sum_{m=0}^1 \sum_{n=0}^1 \sum_{p=0}^1 \sum_{q=0}^1 \sum_{r=0}^1 \sum_{s=0}^1 B_{mn}^{(i)} B_{pq}^{(j)} B_{rs}^{(k)} R_{mpr}^{(D)}(C_{nqs}) \quad (17)$$

where the i -entry of D_{1pr} , the j -entry of D_{m1r} , and the k -entry of D_{mp1} are all 1, while the others are zero, indicating the appropriate first, cross, or triple derivative. The i -entries of C_{0qs} and C_{1qs} , the j -entries of C_{n0s} and C_{n1s} , and the k -entries of C_{nq0} and C_{nq1} are equal to the corresponding entries of $START$ and END :



For Lagrange interpolation the limit of the first summation in the projector in Eq. (15) is 0, leaving m with a value only of 0. Incomplete Hermite interpolation, involving the specification of the derivative at only one end can be represented by making the limits on the second summation in the projector dependent on m . All of these forms can be incorporated in the notation

$$P^i = \sum_{m=0}^M \sum_{n=N_m^{(1)}}^{N_m^{(2)}} B_{mn}^{(1)}(\xi) \underline{R}^{(D_m)}(\underline{C}_n) \quad (18)$$

where $M=0$ for Lagrange and $M=1$ otherwise, and where $N_0^{(1)} = 0$ and $N_0^{(2)} = 1$. With the derivative specified at only the first end, the other limits are $N_1^{(1)} = N_1^{(2)} = 0$, while with the specification at only the second end they are $N_1^{(1)} = N_1^{(2)} = 1$. For complete Hermite, the limits are $N_1^{(1)} = 0$ and $N_1^{(2)} = 1$, of course.

In multiple dimensions, the product projectors in Eq. (16) and Eq. (17) may involve different forms of interpolation in the different directions, with these products written as

$$P^i P^j = \sum_{m=0}^M \sum_{n=N_m^{(1)}}^{N_m^{(2)}} \sum_{p=0}^P \sum_{q=Q_p^{(1)}}^{Q_p^{(2)}} B_{mn}^{(i)} B_{pq}^{(j)} \underline{R}^{(D_{mp})}(\underline{C}_{nq}) \quad (19)$$

and

$$p^i p^j p^k = \sum_{m=0}^M \sum_{n=N_m^{(1)}}^{N_m^{(2)}} \sum_{p=0}^P \sum_{q=Q_p^{(1)}}^{Q_p^{(2)}} \sum_{r=0}^R \sum_{s=S_r^{(1)}}^{S_r^{(2)}} B_{mn}^{(1)} B_{pq}^{(j)} B_{rs}^{(k)} \underline{R}^{(D_{mpr})} (C_{nqs}) \quad (20)$$

with the limits set individually for the form used in each direction.

The blending functions for all three forms are as follows:

LAGRANGE (Points only specified at each end)

$$B_{00}^{(1)} = 1 - \xi^1$$

$$B_{01}^{(1)} = \xi^1$$

HERMITE1 (Points at both ends, derivative at first end)

$$B_{00}^{(1)} = 1 - \xi^{1^2}$$

$$B_{01}^{(1)} = \xi^{1^2}$$

$$B_{10}^{(1)} = \xi^1 (1 - \xi^1)$$

HERMITE2 (Points at both ends, derivative at second end)

$$B_{00}^{(1)} = (1 - \xi^1)^2$$

$$B_{01}^{(1)} = \xi^1 (2 - \xi^1)$$

$$B_{11}^{(1)} = \xi^1 (\xi^1 - 1)$$

HERMITE (Points and derivatives at both ends)

$$B_{00}^{(1)} = (1 + 2\xi^1)(1 - \xi^1)^2$$

$$B_{01}^{(i)} = (3-2\xi^i)\xi^{i^2}$$

$$B_{10}^{(i)} = (1-\xi^i)^2\xi^i$$

$$B_{11}^{(i)} = (\xi^i-1)\xi^{i^2}$$

The argument ξ^i of these blending functions is defined to vary from 0 to 1 from the first to the second end. Therefore, with LINEAR blending functions, this ξ^i is given by

$$\xi^i = \frac{C(i)-START(i)}{END(i)-START(i)} \quad (21)$$

where the $C(i)$ on the right is the curvilinear coordinate varying by unit increments from $START(i)$ to $END(i)$ from the first to the second end. With ARC blending functions, ξ^i is the relative arc length from the first end.

Specification of the spacing, $\Delta^{(i)}r$, at an end amounts to specifying:

$$\frac{dr}{dC(i)} = \frac{\Delta^{(i)}r}{\Delta C(i)} = \Delta^{(i)}r$$

since $C(i)$ varies by unit increments. The r_{ξ^i} for the interpolation is then given by

$$r_{\xi^i} = \frac{dr}{d\xi^i} = \frac{dr}{dC(i)} \frac{dC(i)}{d\xi^i} = \frac{dC(i)}{d\xi^i} \Delta^{(i)}r$$

For LINEAR blending functions,

$$\frac{dC(i)}{d\xi^i} = END(i) - START(i)$$

while for ARC blending functions, with A for relative arc length,

$$\frac{dC(i)}{d\xi^1} = \frac{dC(i)}{dA} \frac{dA}{d\xi^1} = \frac{1}{\Delta^{(i)}_A} \frac{dA}{d\xi^1} = \frac{1}{\Delta^{(i)}_A}$$

since relative arc length also varies on the range 0-1 and hence can be interpreted directly as ξ^1 . Then for LINEAR blending functions,

$$r_{\xi^1} = [END(i) - START(i)] \Delta^{(i)}_r \quad (22)$$

while for ARC blending functions

$$r_{\xi^1} = \frac{\Delta^{(i)}_r}{\Delta^{(i)}_A} \quad (23)$$

In the code, the following notation is used:

m,p,r → NF1, NF2, NF3 or NF(1), NF(2), NF(3)

n,q,s → NE1, NE2, NE3 or NE(1), NE(2), NE(3)

M,P,R → LF(form)

$N_m^{(1)}$, $Q_p^{(1)}$, $S_r^{(1)}$ → LE(1, m, form) etc.

$B_{mn}^{(i)}(\xi) \rightarrow BLEN(1, m, n, C(1), C(2), C(3))$

$R_{mpqr}^{(D)}(C_{nqs}) \rightarrow RB(3, m, p, r, NDX(CC(n,1), CC(q,2), CC(s,3)))$

where CC(0, i) is START(i), or C(i) and CC(1,i) is either END(i) or C(i)

$\xi^1 \rightarrow FACIN(1, C(1), C(2), C(3))$

APPENDIX B

ELLIPTIC GENERATION SYSTEM

Laplace system

The most simple elliptic partial differential system is the Laplace system (cf. Ref. 1):

$$\nabla^2 \xi^i = 0 \quad (i = 1, 2, 3) \quad (1)$$

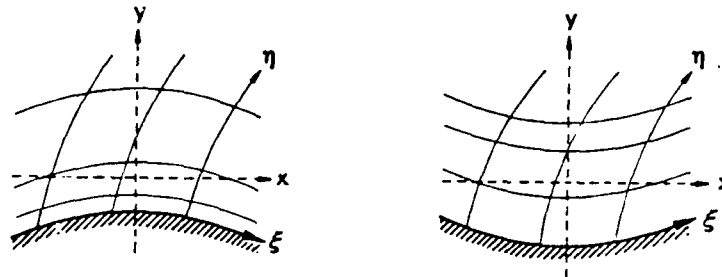
This generation system guarantees a one-to-one mapping on general closed boundaries. These equations can, in fact, be obtained from the Euler equations for the minimization of the integral

$$I = \iiint \sum_{i=1}^3 |\nabla \xi^i|^2 dV \quad (2)$$

Since the coordinate lines are located at equal increments of the curvilinear coordinate, the quantity $|\nabla \xi^i|$ can be considered a measure of the grid point density along the coordinate line on which ξ^i varies, i.e., ξ^i must change rapidly in physical space where grid points are clustered. Minimization of this integral thus leads to the smoothest coordinate line distribution over the field.

With this generating system the coordinate lines will tend to be equally spaced in the absence of boundary curvature because of the strong smoothing effect of the Laplacian, but will become more closely spaced over convex boundaries, and less so over concave boundaries, as illustrated below. (In this and other illustrations and applications in two dimensions, ξ^1 and ξ^2 will be denoted ξ and η , respectively, ξ

and x_2 and y will be used for x_1 , and x_2 .)



In the left figure we have $\eta_{xx} > 0$ because of the convex (to the interior) curvature of the lines of constant η (η -lines). Therefore it follows that $\eta_{yy} < 0$, and hence the spacing between the η -lines must increase with y . The η -lines thus will tend to be more closely spaced over such a convex boundary segment. For concave segments, illustrated in the right figure, we have $\eta_{xx} < 0$, so that η_{yy} must be positive, and hence the spacing of the η -lines must decrease outward from this concave boundary.

Poisson system

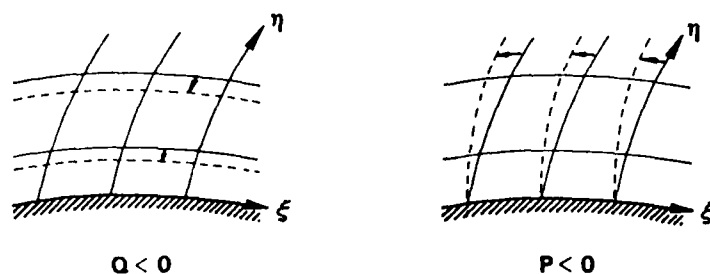
Control of the coordinate line distribution in the field can be exercised by generalizing the elliptic generating system to Poisson equations:

$$\nabla^2 \xi^i = p^i \quad (i = 1, 2, 3) \quad (3)$$

in which the "control functions" p^i can be fashioned to control the spacing and orientation of the coordinate lines.

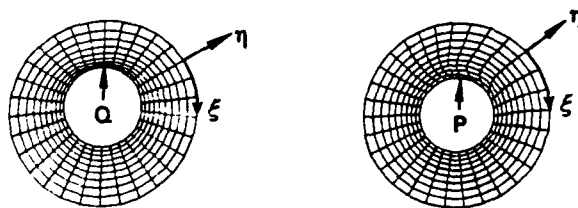
Considering the equation $\nabla^2 \eta = Q$ and the figures above ($p^1 = P$ and $p^2 = Q$ in the illustrations here), since a negative value of the control function would tend to make η_{yy} more negative, it follows that ne-

gative values of Q will tend to cause the coordinate line spacing in the cases shown above to increase more rapidly outward from the boundary. Generalizing, negative values of the control function Q will cause the η -lines to tend to move in the direction of decreasing η , while negative values of P in $\nabla^2 \xi = P$ will cause ξ -lines to tend to move in the direction of decreasing ξ . These effects are illustrated below for an η -line boundary:



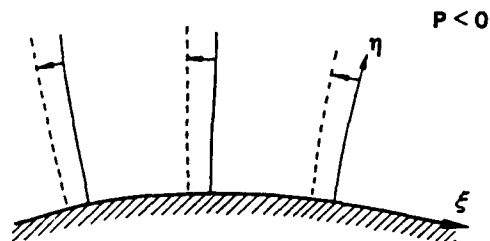
With the boundary values fixed, the ξ -lines here cannot change the intersection with the boundary. The effect of the control function P at the boundary in this case is to change the angle of intersection, causing the ξ -lines to lean in the direction of decreasing ξ .

These effects are illustrated in the following figures:



Here the ξ -lines are radial and the η -lines are circumferential. In the left illustration the control function Q is locally non-zero near a portion of the inner boundary as indicated, so the η -lines move closer to that portion of the boundary while in the right figure, P is locally

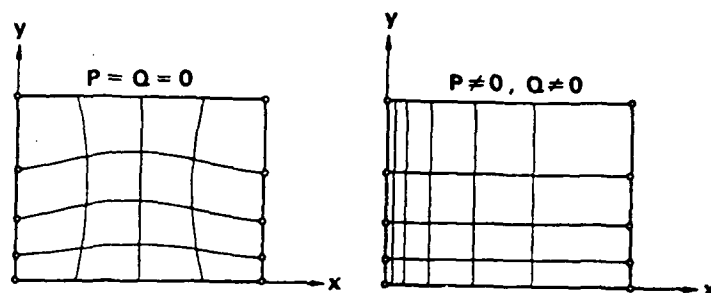
non-zero, resulting in a change in intersection angle of the ξ -lines with that portion of the boundary. If the intersection angle, instead of the point location, on the boundary is specified, so that the points are free to move along the boundary, then the ξ -lines would move toward lines with lower values of ξ :



In general, a negative value of the Laplacian of one of the curvilinear coordinates causes the lines on which that coordinate is constant to move in the direction in which that coordinate decreases. Positive values of the Laplacian naturally result in the opposite effect.

Effect of boundary point distribution

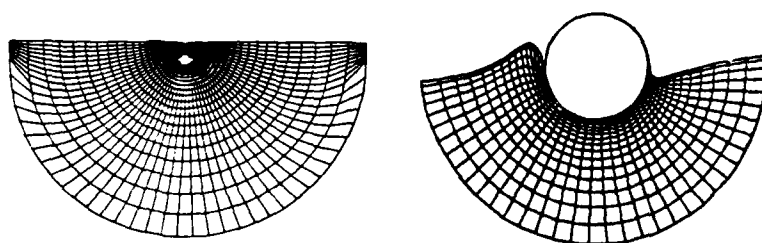
Because of the strong smoothing tendencies that are inherent in the Laplacian operator, in the absence of the control functions, i.e., with $P^i = 0$, the coordinate lines will tend to be generally equally spaced away from the boundaries regardless of the boundary point distribution. For example, the simple case of a coordinate system comprised of horizontal and vertical lines in a rectangular physical region, (cf. the right figure below) cannot be obtained as a solution of Eq. (3) with $P=Q=0$ unless the boundary points are equally spaced.



With $\xi_{yy} = \eta_{xx} = 0$, Eq. (3) reduces to

$$\xi_{xx} = P, \quad \eta_{yy} = Q$$

and thus P and Q cannot vanish if the point distribution is not uniform on the horizontal and vertical boundaries, respectively. With $P=Q=0$ the lines tend to be equally-spaced away from the boundary. These effects are illustrated further in the figures below. Here the control functions are zero in the left figure.



Although the spacing is not uniform on the semi-circular outer boundary in this figure, the angular spacing is essentially uniform away from the boundary. By contrast, nonzero control functions in the right figure, evaluated from the boundary point distribution, cause the field spacing to follow that on the boundary. Thus, if the coordinate lines in the interior of the region are to have the same general spacing as the point distributions on the boundaries which these lines connect, it

is necessary to evaluate the control functions to be compatible with the boundary point distribution. This evaluation of the control functions from the boundary point distribution is discussed in Appendix C.

General Poisson-type systems

If a curvilinear coordinate system, $\bar{\xi}^i$ ($i = 1, 2, 3$), which satisfies the Laplace system

$$\nabla^2 \bar{\xi}^i = 0 \quad (i = 1, 2, 3)$$

is transformed to another coordinate system, ξ^i ($i = 1, 2, 3$), then the new curvilinear coordinates, ξ^i , satisfy the inhomogeneous elliptic system (cf. Ref. [1])

$$\nabla^2 \xi^i = p^i \quad (i = 1, 2, 3) \quad (4)$$

where

$$p^i = \sum_{j=1}^3 \sum_{k=1}^3 g^{jk} p_{jk}^i \quad (5)$$

where the g^{jk} are the elements of the contravariant metric tensor, and with the p_{jk}^i defined by the transformation from $\bar{\xi}^i$ to ξ^i :

$$P_{jk}^i = \sum_{m=1}^3 \sum_{n=1}^3 \frac{\partial \bar{\xi}^m}{\partial \xi^j} \frac{\partial \bar{\xi}^n}{\partial \xi^k} \frac{\partial^2 \xi^i}{\partial \bar{\xi}^m \partial \bar{\xi}^n} \quad (6)$$

(It may be noted that if the subsequent transformation is one-dimensional, i.e., if $\partial \bar{\xi}^1 / \partial \xi^j = \delta_j^1 \partial \bar{\xi}^1 / \partial \xi^j$ then only the three functions P_{ii}^1 , with $i = 1, 2, 3$, are nonzero.)

These results show that a grid with lines concentrated by applying a subsequent transformation (often called a "stretching" transformation) to a grid generated as the solution of the Laplace system could have been generated directly as the solution of the Poisson system (4) with appropriate "control functions", P_{jk}^i , derived from the subsequent concentrating transformation according to Eq. (6). Therefore, it is appropriate to adopt this Poisson system (4) as the generation system, but with the control functions specified directly rather than through a subsequent transformation.

Thus an appropriate generation system can be defined by Eqs. (4) and (5):

$$\nabla^2 \xi^i = \sum_{j=1}^3 \sum_{k=1}^3 g^{jk} P_{jk}^i \quad (i = 1, 2, 3) \quad (7)$$

with the control functions, P_{jk}^i , considered to be specified. The basis of the generation system (7) is that it produces a coordinate system that corresponds to the subsequent application of a stretching trans-

formation to a coordinate system generated for maximum smoothness. From Eq. (6), the three control functions, P_{1i}^i ($i = 1, 2, 3$), correspond to one-dimensional stretching in each coordinate direction and thus are the most important of the control functions. In applications, in fact, the other control functions have been taken to be zero, i.e., $P_{jk}^i = \delta_j^i \delta_k^i P_i^i$, so that the generation system becomes

$$\nabla^2 \xi^i = g^{ii} P_i^i \quad (i = 1, 2, 3) \quad (8)$$

It may be noted that Eq. (7) can be written as

$$\nabla^2 \xi^i = \sum_{j=1}^3 \sum_{k=1}^3 P_{jk}^i (\nabla \xi^j \cdot \nabla \xi^k) = 0 \quad (9)$$

Actual computation is to be done in the rectangular transformed field where the curvilinear coordinates, ξ^i , are the independent variables, with the Cartesian coordinates, x_i , as dependent variables. The transformation of Eq. (9) is

$$\sum_{i=1}^3 \sum_{j=1}^3 g^{ij} (r_{\xi^i \xi^j} + \sum_{k=1}^3 P_{ij}^k r_{\xi^k}) = 0 \quad (10)$$

This then is the quasi-linear elliptic partial differential equation which can be solved to generate the coordinate system. (In computation, the Jacobian squared, g , can be omitted from the evaluation of the metric coefficients, g^{ij} , in this equation since it would cancel anyway.)

Generation System

The elliptic grid generation system used in this code is

$$\sum_{m=1}^3 \sum_{n=1}^3 g^{mn} r_{\xi^m \xi^n} + \sum_{n=1}^3 g^{nn} P_n r_{\xi^n} = 0 \quad (11)$$

where the g^{mn} are the elements of the contravariant metric tensor:

$$g^{mn} = \nabla \xi^m \cdot \nabla \xi^n$$

These elements are more conveniently expressed in terms of the elements of the covariant metric tensor, g_{mn} :

$$g_{mn} = r_{\xi^m} \cdot r_{\xi^n}$$

which can be calculated directly. Thus

$$g^{mn} = (g_{ik} g_{jl} - g_{il} g_{jk}) / g$$

(m,i,j) cyclic, (n,k,l) cyclic

where g , the square of the Jacobian, is given by

$$g = \det |g_{mn}| = r_{\xi^1} \cdot (r_{\xi^2} \times r_{\xi^3})$$

In these relations, r is the Cartesian position vector of a grid point ($r = ix + jy + kz$), and the ξ^i ($i=1,2,3$) are the three curvilinear coordinates. The P_n are the three 'control functions' which serve to control the spacing and orientation of the grid lines in the field.

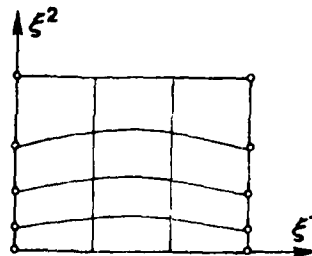
APPENDIX C

CONTROL FUNCTIONS

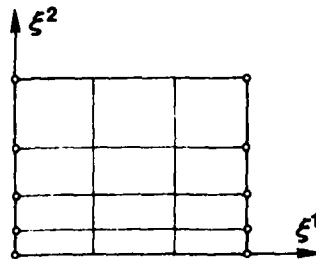
Negative values of the control function P_n cause grid lines on which ξ^n is constant to tend to move in the direction of decreasing ξ^n , and this feature can be used to concentrate grid lines near other grid lines and/or points or in certain regions of physical space. However, a more automatic procedure is to determine the control functions so as to reflect the boundary point spacing into the field. The details of how this is done in the present code are discussed later in this appendix, but there follows first a less general, but more immediately enlightening, explanation.

General Development

Consider first a physical rectangle with equally spaced points on the horizontal sides but the same unequal spacing on the two vertical sides. With no control functions, i.e., $P_n=0$, Eq. (B-11) of Appendix B will produce a grid that attempts to be equally spaced away from the unequal spacing on the vertical sides:



A grid of parallel lines for this configuration, reflecting the unequal spacing on the boundaries:



can only be produced from Eq. (B-11) with $P_2 \neq 0$ (taking ξ^1 to vary on the horizontal sides and ξ^2 to vary on the vertical sides).

The proper values of P_2 needed to accomplish this are determined by evaluating Eq. (B-11) one-dimensionally on the vertical sides, with the result

$$y_{\xi^2 \xi^2}^2 + P_2 y_{\xi^2}^2 = 0$$

so that

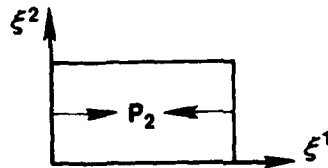
$$P_2 = - \frac{y_{\xi^2 \xi^2}^2}{y_{\xi^2}^2} \quad (1)$$

If there are J points on the vertical sides, and I points on the horizontal sides, the control function P_2 on the vertical sides then can be evaluated from the point distribution thereon as

$$P_2(1, j) = - \frac{[y(1, j+1) - 2y(1, j) + y(1, j-1)]}{\frac{1}{2} [y(1, j+1) - y(1, j-1)]}$$

for $j=2, 3, \dots, J-1$, with an analogous equation with I for the first argu-

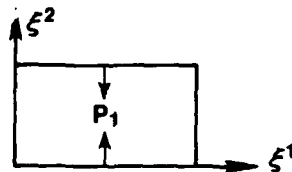
ment. The values of P_2 in the interior of the region then can be interpolated between the two vertical sides:



A similar evaluation of the other control function, P_1 , on the two horizontal sides from

$$P_1 = - \frac{x_1^1 \xi_1^1}{\xi_1^1} \quad (2)$$

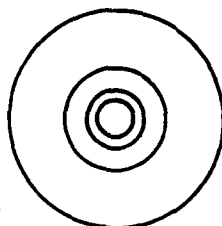
produces a zero in the present case with equal spacing on these sides. In the case of unequal spacing, the values of P_1 in the interior of the region would be evaluated by interpolation between the values on the two horizontal sides:



With the control functions evaluated in this manner, Eq. (B-11), will produce a grid composed of parallel straight lines for this boundary configuration, thus reflecting the boundary point spacing into the field.

Now consider an O-type grid with two concentric circular boundaries and equally spaced points around the circles. Because of its inherent tendency to cause the grid lines to move closer to convex

boundaries, Eq. (B-11) with no control functions, will produce a grid with unequal radial spacing of the circumferential lines:



In this case, evaluation of Eq. (1) with

$$x(r, \theta) = r(\xi^2) \cos \theta$$

$$y(r, \theta) = r(\xi^2) \sin \theta$$

yields the equation

$$P_2 = -\frac{r^2 \xi^2}{r \xi^2} + \frac{r \xi^2}{r} \quad (3)$$

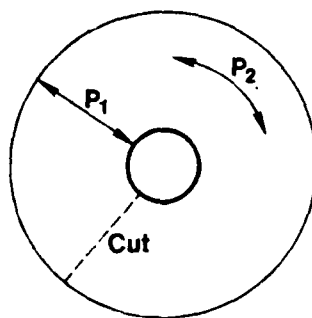
Thus, in order to produce a specified radial distribution of lines, the control function P_2 must be evaluated from Eq. (3) using the given distribution $r(\xi^2)$. Now the computational field here has its two vertical sides corresponding to a cut between the two circular boundaries in the physical field:



Therefore with the specified radial distribution placed on the two vertical sides of the computational field, the control function P_2 can be evaluated from Eq. (3) on these sides as

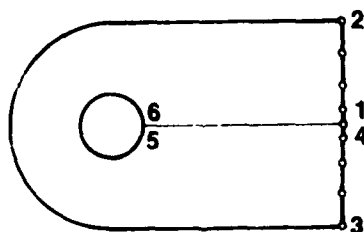
$$P_2(1,j) = - \frac{[r(1,j+1) - 2r(1,j) + r(1,j-1)]}{\frac{1}{2} [r(1,j+1) + r(1,j-1)]} + \frac{\frac{1}{2} [r(1,j+1) + r(1,j-1)]}{r(1,j)} \quad (4)$$

and a similar equation with I as the first argument. The interior values then can be determined by interpolation between the two vertical sides as before. Note that this amounts to interpolation in the circumferential direction in the physical plane, as indicated in the figure below. Again, equal spacing around the circles produces a zero value of the other control function, P_1 . In general, P_1 would be evaluated on the two circles and interpolated between the two horizontal sides in the computational region, i.e., between the two circles in the physical field:

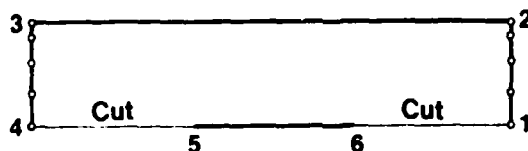


The second term in Eq. (3) arises from the curvature of the boundary, and the denominator is the local radius of curvature of the grid line that is to pass through the point where the control function is being evaluated. This term acts to reduce the magnitude of the control function in order to allow for the natural tendency of the grid lines to move toward convex boundaries. Since the lines tend to concentrate near the inner circle even with zero control functions, the use of the first term alone in Eq. (3), (in analogy with the flat boundary case, i.e., Eq. (1)), would produce a stronger concentration of lines near the inner circle than was intended.

Finally, consider a C-type grid:



with the computational region



If now the control function P_2 is evaluated on lines 1-2 and 4-3 in the physical field from Eq. (1), and the interior values are interpolated between the two vertical sides in the computational field, the resulting control function, while serving well over the right portion of the physical field, will be too strong over the inner body where the line curvature is not zero. The use of Eq. (3) on the lines 1-2 and 4-3 would be no better since the r in the denominator is to be interpreted as the local radius of curvature of the crossing line and hence is infinite on these lines so that the second term in Eq. (3) vanishes.

This situation can be remedied by interpolating for the local radius of curvature in Eq. (3) between the inner and outer boundaries in the physical field, i.e., between the horizontal sides in the computational region. However, since the ξ^2 derivatives in Eq. (3) must still be evaluated on the vertical sides it is necessary to separate Eq. (3) into three pieces:

$$P_2 = -\frac{r}{\xi^2} \frac{\xi^2 \xi^2}{r^2} + \frac{r}{r} \frac{\xi^2}{r} = A_2 + \frac{s_2}{\rho_2} \quad (5)$$

where A_2 is the contribution from the rate of change of the arc length spacing,

$$A_2 = -\frac{r}{\xi^2} \frac{\xi^2 \xi^2}{r^2}$$

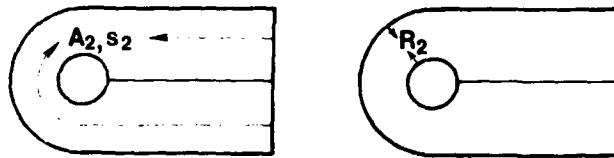
s_2 in the arc length spacing,

$$s_2 = r_2^2$$

and ρ_2 is the radius of curvature,

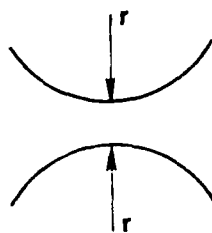
$$\rho_2 = r$$

Now the arc length contributions, A_2 and s_2 , are evaluated on the vertical sides of the computational region, while the radius of curvature, ρ_2 , is evaluated on the two horizontal sides. The control function in the interior then is evaluated by interpolating the arc length contributions between the vertical sides interpolating the radius of curvature between the horizontal sides, and then evaluating P_2 from Eq. (5) using these interpolated values:



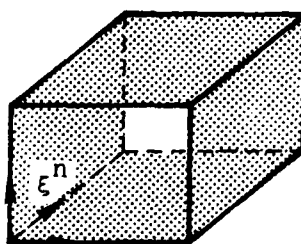
Note that this procedure supplies a finite radius of curvature over the inner body, thus reducing the control function appropriately in this region.

A problem arises, however, when the radius of curvature is of opposite sign on the two boundaries between which it is interpolated:

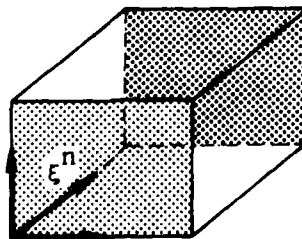


since then the interpolation will produce a zero value at some point in between, and at such a point the second term of Eq. (5) is finite. Special measures are then necessary as discussed in Appendix G.

Although the exact equations for the general case are more complicated, they still may be separated into terms arising from the spacing along a boundary and terms arising from the local curvature of the crossing lines, with the spacing terms, A_n and s_n , for the control function P_n being interpolated between the four sides on which ξ^n varies:



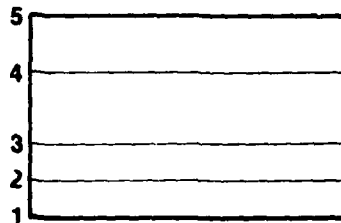
and the curvature term, ρ_n , for P_n interpolated between the two sides on which P_n is constant:



The control function is then evaluated from

$$P_n = A_n + \frac{s_n}{\rho_n} \quad (6)$$

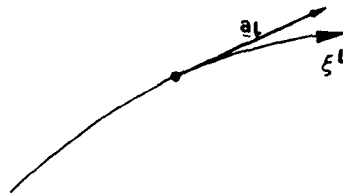
The question that then arises is whether the transfinite interpolation for the spacing terms and the curvature term should use linear blending functions or blending functions based on physical arc length. The former (linear) amounts to interpolating in terms of the curvilinear coordinate, while the latter (arc) amounts to interpolation with respect to the physical distance. For example, on the grid illustrated below:



interpolation with linear blending functions would produce a value on line 3 that is the average of that on lines 1 and 5, while with arc blending functions the value produced on line 3 would be closer to that on line 1. The code uses linear blending functions for the arc length contributions, A and s , and blending functions based on arc length for the radius of curvature.

Evaluation along a coordinate line

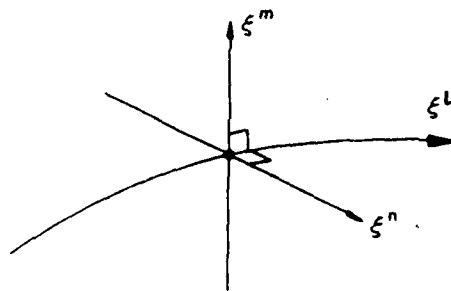
The projection of Eq. (B-11) along a coordinate line on which ξ^k varies is found by forming the dot product of this equation with the covariant base vector $\underline{a}_k = \underline{r}_{\xi^k}$, which is tangent to the line.



Thus we have

$$\sum_{i=1}^3 \sum_{j=1}^3 g^{ij} r_{\xi^l} \cdot r_{\xi^i \xi^j} + \sum_{k=1}^3 g^{kk} p_k r_{\xi^l} \cdot r_{\xi^k} = 0 \quad (7)$$

Now assume for the moment that the two coordinate lines crossing the coordinate line of interest do so orthogonally. Then on this line we have



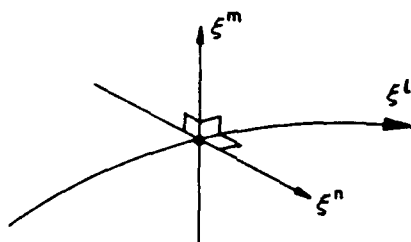
and

$$r_{\xi^l} \cdot r_{\xi^k} = a_l \cdot a_k = g_{lk} = \delta_{lk} g_{ll}$$

which leads to an explicit equation for P_l on the coordinate line of interest:

$$P_l = \frac{-1}{g^{ll}g_{ll}} r_{\xi^l} \cdot \sum_{i=1}^3 \sum_{j=1}^3 g^{ij} r_{\xi^i \xi^j} \quad (8)$$

If it is further assumed for the moment that the two coordinate lines crossing the coordinate line of interest are also orthogonal to each other, i.e., complete orthogonality on the line of interest,



we have on this line $g^{ij} = \delta_{ij}g^{ii}$ and $g_{ij} = \delta_{ij}g_{ii}$. Also,

$$g^{ll} = \frac{1}{g} (g_{mm}g_{nn} - g_{mn}^2) = \frac{1}{g} g_{mm}g_{nn} \quad (l, m, n, \text{ cyclic})$$

since $m \neq n$. But also $g = g_{ll}g_{mm}g_{nn}$ so that $g^{ll}g_{ll} = 1$. Then Eq. (8) becomes

$$P_l = - \sum_{i=1}^3 \frac{1}{g_{ii}} r_{\xi^l} \cdot r_{\xi^i \xi^i} \quad (l = 1, 2, 3) \quad (9)$$

which can also be written as

$$P_l = \sum_{i=1}^3 \frac{1}{g_{ii}} a_i \cdot (a_i)_{\xi^l} \quad (10)$$

The derivative of arc length along the coordinate line on which ξ^l varies is

$$s_{\xi^l} = |r_{\xi^l}| \quad (11)$$

Then

$$s_{\xi^l \xi^l} = \frac{r_{\xi^l} \cdot r_{\xi^l \xi^l}}{|r_{\xi^l}|} \quad (12)$$

so that the logarithmic derivative of arc length along this coordinate line is given by

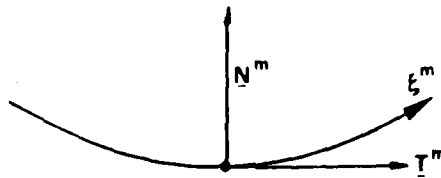
$$S_l = \frac{s_{\xi^l \xi^l}}{s_{\xi^l}} = \frac{r_{\xi^l} \cdot r_{\xi^l \xi^l}}{|r_{\xi^l}|^2} = \frac{r_{\xi^l} \cdot r_{\xi^l \xi^l \xi^l}}{g_{ll}} \quad (13)$$

which is exactly the $i=l$ term in the summation in Eq. (9).

The unit tangent to a coordinate line on which ξ^m varies is

$$T^m = \frac{r_{\xi^m}}{|r_{\xi^m}|} = \frac{a_m}{|a_m|} \quad (14)$$

and the derivative of this unit tangent with respect to arc length is a vector that is normal to this line, the magnitude of which is the curvature, K , of the line. The unit vector in this normal direction is the principal normal, N , to the line.



Thus, using Eq. (11),

$$K^m_{\underline{N}^m} = (\underline{T}^m)_S = (\underline{T}^m)_{\xi^m} \xi^m_S = \frac{(\underline{T}^m)_{\xi^m}}{|\underline{r}_{\xi^m}|} \quad (15)$$

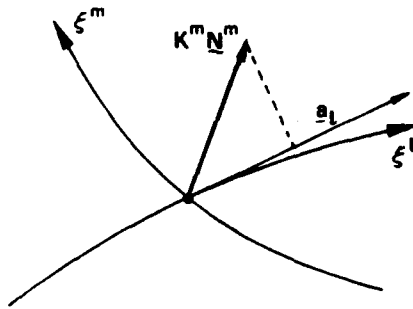
Then

$$\begin{aligned} K^m_{\underline{N}^m} &= \frac{1}{|\underline{r}_{\xi^m}|} \left(\frac{\underline{r}_{\xi^m}}{|\underline{r}_{\xi^m}|} \right)_{\xi^m} \\ &= \frac{|\underline{r}_{\xi^m}|^2 \underline{r}_{\xi^m \xi^m} - (\underline{r}_{\xi^m} \cdot \underline{r}_{\xi^m \xi^m}) \underline{r}_{\xi^m}}{|\underline{r}_{\xi^m}|^4} \end{aligned} \quad (16)$$

so that the curvature is

$$K^m = \frac{1}{|\underline{r}_{\xi^m}|^3} \left[|\underline{r}_{\xi^m}|^2 |\underline{r}_{\xi^m \xi^m}|^2 - (\underline{r}_{\xi^m} \cdot \underline{r}_{\xi^m \xi^m})^2 \right] \quad (17)$$

The component of $K^m_{\underline{N}^m}$ along the coordinate line on which ξ^l varies



is given by

$$(\underline{K}_{\underline{N}^m})^{(l)} = \underline{K}_{\underline{N}^m} \frac{\underline{r}_{\xi^l}}{|\underline{r}_{\xi^l}|} = \frac{\underline{r}_{\xi^l} \cdot \underline{r}_{\xi^m}}{|\underline{r}_{\xi^l}| |\underline{r}_{\xi^m}|} \quad (18)$$

since $\underline{r}_{\xi^l} \cdot \underline{r}_{\xi^m} = 0$ for $m \neq l$. Then the two terms of the summation in Eq. (9) for which $m \neq l$ can be written as

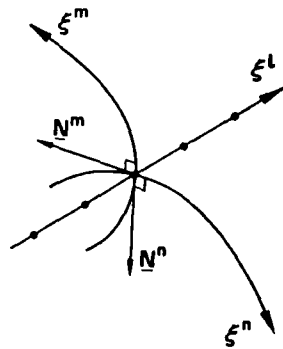
$$\frac{\underline{r}_{\xi^l} \cdot \underline{r}_{\xi^m}}{g_{mm}} = |\underline{r}_{\xi^l}| (\underline{K}_{\underline{N}^m})^{(l)} = \sqrt{g_{ll}} (\underline{K}_{\underline{N}^m})^{(l)} \quad (m \neq l) \quad (19)$$

Thus Eq. (9) can be written

$$P_l = -S_l + \sqrt{g_{ll}} [(\underline{K}_{\underline{N}^m})^{(l)} + (\underline{K}_{\underline{N}^n})^{(l)}] \quad (l=1,2,3) \quad (20)$$

where (l,m,n) are cyclic. This is the equation used in CONLINE. The separate evaluation of the arc length and curvature contributions in CONSURR and CONLINR also are based on this equation. The arc length in the expressions in (20) for the control function P_l along the coordinate line on which ξ^l varies can be determined entirely from the grid

point distribution on the line. The other two terms in P_l , however, involve derivatives off this line and therefore must either be determined by specifications of the components of the curvatures \underline{KN} of the crossing lines along the line of interest or by interpolation between values evaluated on coordinate surfaces intersecting the ends of this line (cf. Ref. 11).



If it is assumed that the curvatures of these crossing lines vanish on the coordinate line of interest, then the last two terms in Eq. (20) are zero, and the control function becomes simply

$$P_l = - S_l \quad (21)$$

and then can be evaluated entirely from the specified point distribution on the coordinate line of interest.

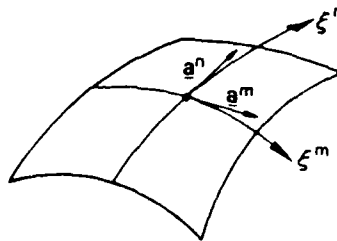
The neglect of the curvature terms, however, is ill-advised since the elliptic system already has a strong tendency to concentrate lines over a convex boundary, as has been noted in Appendix B. Therefore neglect of the curvature terms will result in control functions which will produce a stronger concentration than intended over convex boundaries (and weaker over concave). When interpolation from the end points is used to determine the curvature term, the entire term (\underline{KN})

should be interpolated, since individual interpolation of the vectors r_{ξ^l} and r_{ξ^m} can give an inappropriate value for the dot product.

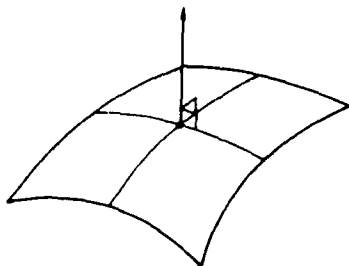
It should be noted that the assumptions of orthogonality, and perhaps vanishing curvature, that were made in the course of the development of these expressions for the control functions on a coordinate line are not actually enforced on the resulting coordinate system, but merely served to allow some reasonable relations for these control functions corresponding to a specified point distribution on a coordinate line to be developed. This should not be considered a source of error since the control functions are arbitrary in the generation system (B-11).

Evaluation on a coordinate surface

In a similar fashion, expressions for the control functions on a coordinate surface on which ξ^l is constant can be obtained from the projections of Eq. (B-11) along the two coordinate lines lying on the surface, i.e., the lines on which ξ^m and ξ^n vary, (l, m, n) being cyclic.



These projections are given by Eq. (7) with l replaced by m and n , respectively. If it is assumed for the moment that the coordinate line crossing the coordinate surface of interest is orthogonal to the surface



then $r_{\xi l} \cdot r_{\xi m} = r_{\xi l} \cdot r_{\xi n} = 0$, so that P_l is removed from both of these two equations to yield the equation

$$\sum_{i=1}^3 \sum_{j=1}^3 g^{ij} r_{\xi m} \cdot r_{\xi i \xi j} + g^{mm} g_{mm} P_m + g^{nn} g_{nn} P_n = 0$$

and an analogous equation with m and n interchanged. Solution of these two equations for P_m and P_n then yields

$$P_m = \frac{(g_{nn} r_{\xi m} - g_{mn} r_{\xi n}) \cdot \sum_{i=1}^3 \sum_{j=1}^3 g^{ij} r_{\xi i \xi j}}{g^{mm} (g_{mm} g_{nn} - g_{mn}^2)} \quad (22)$$

with an analogous equation for P_n with m and n interchanged. Since $g_{lm} = g_{ln} = 0$ it follows $g^{lm} = g^{ln} = 0$. Therefore only the five terms, ll , mm , nn , mn , nm , are non-zero in the summation. Also

$$g^{mm} = \frac{g_{nn}}{g_{mm} g_{nn} - g_{mn}^2}$$

since here $g = \det |g_{ij}| = g_{ll} (g_{mm} g_{nn} - g_{mn}^2)$. An analogous equation for g^{nn} is obtained by interchanging m and n .

Then Eq. (22) can be rewritten as

$$P_m = - \left(r_{\xi^m} - \frac{g_{mn}}{g_{nn}} r_{\xi^n} \right) \cdot \left(\sum_{i=1}^3 g^{ii} r_{\xi^i \xi^i} + 2g^{mn} r_{\xi^m \xi^n} \right) \quad (23)$$

and an analogous equation for P_n with m and n interchanged. But,

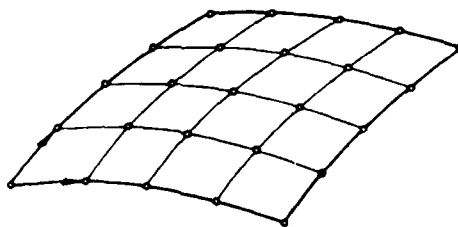
$$g^{ll} = \frac{1}{g_{ll}} \quad \text{and} \quad g^{mn} = \frac{-g_{mn}}{g_{mm}g_{nn} - g_{mn}^2}$$

Therefore

$$\begin{aligned} P_m = & - \frac{1}{g_{ll}} \left(r_{\xi^m} - \frac{g_{mn}}{g_{nn}} r_{\xi^n} \right) \cdot r_{\xi^l \xi^l} \\ & - \left(\frac{r_{\xi^m} - \frac{g_{mn}}{g_{nn}} r_{\xi^n}}{g_{mm}g_{nn} - g_{mn}^2} \right) \\ & \cdot (g_{nn} r_{\xi^m \xi^m} + g_{mm} r_{\xi^n \xi^n} - 2g_{mn} r_{\xi^m \xi^n}) \end{aligned} \quad (24)$$

and the analogous equation with m and n interchanged. This is the equation used in CONSURF.

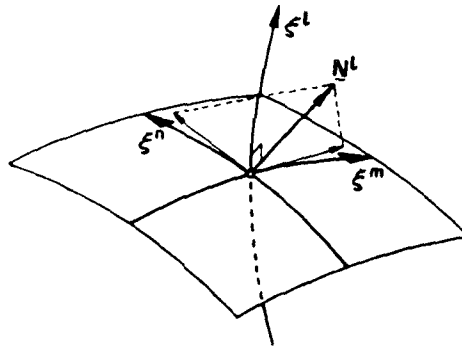
All of the terms, except the first, in the above equations can be evaluated completely from the point distribution on the coordinate surface of interest.



The first term in (24) can be written

$$= \left[\sqrt{g_{mn}} (K^l_{\underline{N}^l})^{(m)} - \frac{g_{mn}}{\sqrt{g_{nn}}} (K^l_{\underline{N}^l})^{(n)} \right] \quad (25)$$

where $(K^l_{\underline{N}^l})^{(m)}$ and $(K^l_{\underline{N}^l})^{(n)}$ are the components of the curvature \underline{KN} for the coordinate line crossing the coordinate surface of interest along the two coordinate lines on the surface.



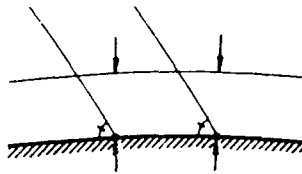
These quantities must be either specified on the surface or interpolated from values evaluated on its intersections with the other coordinate surfaces. If it is further assumed that the curvature of the crossing line vanishes at the surface, then this first term in Eq. (24) vanishes also.

As was noted for the control functions on a line, the curvature terms should not be neglected, however, else the concentration will be stronger than intended over convex boundaries and weaker over concave. Also, it is the entire term \underline{KN} which should be interpolated, not the individual vectors involved, else the dot product can have inappropriate values.

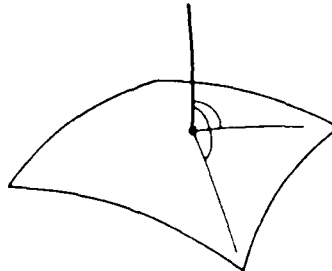
APPENDIX D

ITERATIVE ADJUSTMENT OF CONTROL FUNCTIONS FOR BOUNDARY ORTHOGONALITY

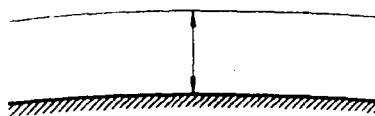
A second-order elliptic generation system allows either the point locations on the boundary or the coordinate line slope at the boundary to be specified, but not both. It is possible, however, to iteratively adjust the control functions in the generation system of the Poisson type discussed above until, not only a specified line slope, but also the spacing of the first coordinate surface off the boundary is achieved, with the point locations on the boundary specified (cf. Ref. 15).



In three dimensions the specification of the coordinate line slope at the boundary requires the specification of two quantities, e.g., the direction cosines of the line with two tangents to the boundary.



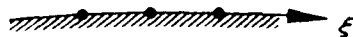
The specification of the spacing of the first coordinate surface off the boundary requires one more quantity,



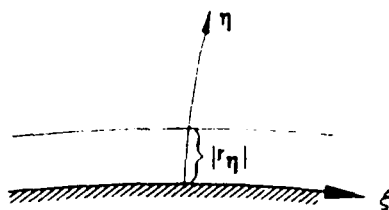
and therefore the three control functions in the system Eq. (B-11) of Appendix B are exactly sufficient to allow these three specified quantities to be achieved, while the one boundary condition allowed by the second-order system provides for the point locations on the boundary to be specified.

General Development

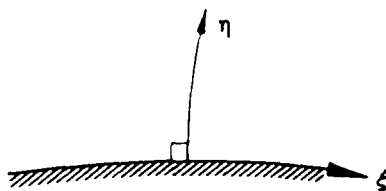
To illustrate this development, an iterative procedure can be constructed for the determination of the control functions in two dimensions as follows (Ref. 15). Consider the generation system given by Eq. (B-11) in two dimensions (with $\xi^1 = \xi$, $\xi^2 = \eta$, $x_1 = x$, and $x_2 = y$ here). On a boundary segment that is a line of constant η we have r_ξ and $r_{\xi\xi}$ known from the specified boundary point distribution



also $|r_\eta|$, the spacing off this boundary, is specified



as is the condition of orthogonality at the boundary, i.e., $\underline{r}_\xi \cdot \underline{r}_\eta = 0$,



But specification of $|\underline{r}_\eta| = \sqrt{x_\eta^2 + y_\eta^2}$, together with the condition $\underline{r}_\xi \cdot \underline{r}_\eta = x_\xi x_\eta + y_\xi y_\eta = 0$ provides two equations for the determination of x_η and y_η in terms of the already known values of the x_ξ and y_ξ . Therefore \underline{r}_η is known on the boundary.

Because of the orthogonality at the boundary, Eq. (B-11) reduces to the following equation on the boundary:

$$|\underline{r}_\eta|^2(\underline{r}_{\eta\eta} + P\underline{r}_\xi) + |\underline{r}_\xi|^2(\underline{r}_{\xi\xi} + Q\underline{r}_\eta) = 0 \quad (1)$$

Dotting \underline{r}_ξ and \underline{r}_η into this equation, and again using the condition of orthogonality, yields the following two equations for the control functions on the boundary:

$$P = - \frac{r_{\xi} \cdot r_{\xi\xi}}{|r_{\xi}|^2} - \frac{r_{\xi} \cdot r_{\eta\eta}}{|r_{\eta}|^2} \quad (2)$$

$$Q = - \frac{r_{\eta} \cdot r_{\eta\eta}}{|r_{\eta}|^2} - \frac{r_{\eta} \cdot r_{\xi\xi}}{|r_{\xi}|^3} \quad (3)$$

All of the quantities in these equations are known on the boundary except $r_{\eta\eta}$. (On a boundary that is a line of constant ξ , the same equations for the control functions result, but now with $r_{\xi\xi}$ the unknown quantity).

The iterative solution thus proceeds as follows:

- (1). Assume values for the control function on the boundary.
- (2). Solve Eq. (1) to generate the grid in the field.
- (3). Evaluate $r_{\eta\eta}$ on η -line boundaries, and $r_{\xi\xi}$ on ξ -line boundaries, from the result of Step (2), using one-sided difference representations. Then evaluate the control functions on the boundary from Eqs. (2) and (3). Evaluate the control functions in the field by interpolation from the boundary values.

Steps (2) and (3) are then repeated until convergence.

Implementation

An iterative solution procedure for the determination of the three control functions for the general three-dimensional case can be constructed as follows. Eq. (C-24) of Appendix C gives the two control functions, P_m and P_n , for a coordinate surface on which ξ^l is constant (l, m, n cyclic) for the case where the coordinate line crossing the surface is normal to the surface. Taking the projection of the generation equation (B-11) of Appendix B on the coordinate line along which ξ^l varies, we have on this same surface,

$$\sum_{i=1}^3 \sum_{j=1}^3 g^{ij} r_{\xi^l}^i \cdot r_{\xi^l \xi^j}^j + g^{ll} g_{ll} / P_l = 0$$

since $g_{lm} = g_{ln}$ are to be zero on the surface. Proceeding as in Appendix C, this equation reduces to

$$P_l = - \frac{1}{g_{ll}} r_{\xi^l}^l \cdot r_{\xi^l \xi^k}^k$$

$$= - \frac{r_{\xi^l}^l}{g_{mm} g_{nn} - g_{mn}^2} \cdot (g_{nn} r_{\xi^l \xi^m}^m + g_{mm} r_{\xi^l \xi^n}^n - 2g_{mn} r_{\xi^l \xi}^m)$$
(4)

Since the coordinate line intersecting the surface is to be normal to the surface, we may write

$$r_{\xi^l}^l = a_l = \sqrt{g_{ll}} \frac{a_m \times a_n}{|a_m \times a_n|}$$

since

$$|\underline{a}_m \times \underline{a}_n|^2 = g_{mm}g_{nn} - g_{mn}^2$$

Eq. (4) can then be written

$$P_l = - \frac{(g_{ll})_{\xi l}}{2g_{ll}} - \frac{\sqrt{g_{ll}}}{(g_{mm}g_{nn} - g_{mn}^2)^{3/2}} (\underline{a}_m \times \underline{a}_n) \cdot [g_{nn}(\underline{a}_m)_{\xi m} + g_{mm}(\underline{a}_n)_{\xi n} - 2g_{mn}(\underline{a}_m)_{\xi n}] \quad (5)$$

With the spacing along the coordinate line intersecting the surface specified at the surface, we have $|\underline{r}_{\xi l}| = \sqrt{g_{ll}}$ known on the surface. Since all the quantities subscripted m or n in Eq. (C-24) and (5) can be evaluated completely from the specified point distribution on the surface, we then have all quantities in these equations for the three control functions on the surface known except for $(g_{ll})_{\xi l}$ and $(\underline{a}_l)_{\xi l}$. These two quantities are not independent, and using Eq. (4), we have

$$(g_{ll})_{\xi^l} = 2a_l \cdot (a_l)_{\xi^l} = \frac{2\sqrt{g_{ll}}}{\sqrt{g_{mm}g_{nn} - g_{mn}^2}} (a_m \times a_n) \cdot (a_l)_{\xi^l} \quad (6)$$

Recall also that $(a_l)_{\xi^l} = r_{\xi^l \xi^l}$.

Therefore, with the control functions in the field determined from the values on the boundary by interpolation, as discussed in the preceeding section, Eq. (C-24) and (5) can be applied to determine the new boundary values of the control functions in terms of the new values of $(a_l)_{\xi^l}$ in an iterative solution. Upon convergence, the coordinate system then will have the coordinate lines intersecting the boundary normally at fixed locations and with the specified spacing on these lines off the boundary.

The three equations for the control functions on the surface can be written in the following notation, taking advantage of common terms,

$$P_m = -B_1 \cdot A$$

$$P_n = -B_2 \cdot A$$

$$P_l = -C \cdot A$$

where

$$A = r_{\xi^l \xi^l} + a$$

$$a = \frac{g_{ll}}{g_{mm}g_{nn} - g_{mn}^2} (g_{nn}r_{\xi^m \xi^m} + g_{mm}r_{\xi^n \xi^n} - 2g_{mn}r_{\xi^m \xi^n})$$

$$B_1 = \frac{1}{g_{ll}} \left(r_{\xi m} - \frac{g_{mn}}{g_{nn}} r_{\xi n} \right)$$

$$B_2 = \frac{1}{g_{ll}} \left(r_{\xi n} - \frac{g_{mn}}{g_{mm}} r_{\xi m} \right)$$

$$C = \frac{r_{\xi m} \times r_{\xi n}}{\sqrt{g_{ll} (g_{mm} g_{nn} + g_{mn}^2)}}$$

Here all quantities except the $r_{\xi l \xi l}$ that appears in A can be evaluated from the boundary point distribution and the specified off-boundary spacing. In previous applications the relations have been applied on the boundary and the control function increments generated at the boundary have been interpolated into the field (Ref. 15). In the present code, these relations are applied on each successive coordinate surface off the boundary, with the off-surface spacing determined by a hyperbolic sine distribution from the spacing specified at the boundary. The control increments are attenuated away from the boundary, and contributions are accumulated from all orthogonal boundary sections.

APPENDIX E

SPLINE

The set of grid points $r(\xi)$, $\xi=1,2,\dots,N$, is splined with the curvilinear coordinate ξ as the parameter. Thus on the interval $\xi_1 \leq \xi \leq \xi_{i+1}$, the curve is given by the cubic polynomial

$$r(u) = \frac{1}{6} s_1 (1-u)^3 + \frac{1}{6} s_{i+1} u^3 + (r_1 - \frac{1}{6} s_1)(1-u) + (r_{i+1} - \frac{1}{6} s_{i+1})u \quad (1)$$

where $s = r_{\xi\xi}$ and $0 \leq u \leq 1$, i.e., $\xi = \xi_1 + u$, and $r_1 = r(\xi_1)$, $r_{i+1} = r(\xi_{i+1})$, etc.

The second derivatives are determined by the tridiagonal solution of

$$s_{i-1} + 4s_i + s_{i+1} = 6(r_{i+1} - 2r_i + r_{i-1}) \quad (2)$$

for $i=2,3,\dots,N-1$. The end values are determined in one of three ways:

(1) zero curvature (natural spline):

$$s_1 = s_N = 0$$

(2) constant curvature (quadratic ends):

$$s_1 = 2s_2 - s_3$$

$$s_N = 2s_{N-1} - s_{N-2}$$

(3) specified slopes:

$$s_1 = 3(r_2 - r_1) + 3g_1 - \frac{1}{2} s_{\alpha 2}$$

$$s_N = -3(r_N - r_{N-1}) + 3g_N - \frac{1}{2} s_{N-1}$$

where $g = r_{\xi}$.

In the first case above, Eq. (2) is replaced by the following equations for $i=2$ and $N-1$:

$$4s_2 + s_3 = 6(r_3 - 2r_2 + r_1)$$

$$4s_{N-1} + s_{N-2} = 6(r_N - 2r_{N-1} + r_{N-2})$$

while in the second case

$$6s_2 = 6(r_3 - 2r_2 + r_1)$$

$$6s_{N-1} = 6(r_N - 2r_{N-1} + r_{N-2})$$

and in the third case

$$\frac{7}{2} s_2 + s_3 = 6(r_3 - 2r_2 + r_1) + 3g_1 + 3(r_2 - r_1)$$

$$\frac{7}{2} s_{N-1} + s_{N-2} = 6(r_N - 2r_{N-1} + r_{N-2}) + 3g_{N-1} + 3(r_N - r_{N-1} + r_{N-2})$$

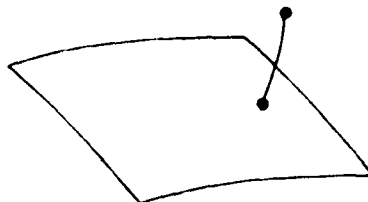
The slopes, $g = r_{\xi}$, are given by

$$g_1 = \frac{1}{2} (r_{i+1} - r_{i-1}) - \frac{1}{12} (s_{i+1} - s_{i-1})$$

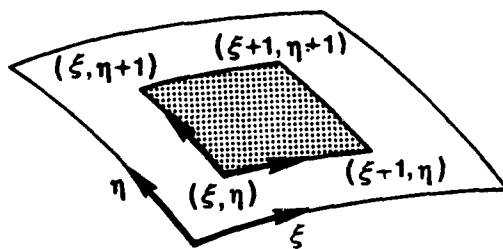
APPENDIX F

NORMAL ON SPLINED SURFACE

The use of Neumann boundary condition requires that a boundary moves over the surface to lie at the foot of surface normal from the adjacent point in the curvilinear direction off the surface:



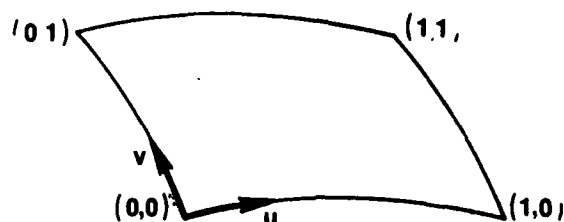
To allow this movement over the surface, a splined surface is created from the initial grid on the surface section involved. This gives a bicubic representation of the surface within each cell of the original surface grid (a tensor-product patch) over which the boundary point can move. The representation of this patch on the cell shown below, with the two curvilinear coordinates on the surface represented by ξ and η ,



is as follows: (cf. Ref. 16):

$$\mathbf{r}(u,v) = [\alpha_0(u) \ \alpha_1(u) \ \beta_0(u) \ \beta_1(u)] \begin{bmatrix} \mathbf{r}(0,0) & \mathbf{r}(0,1) & \mathbf{r}_v(0,0) & \mathbf{r}_v(0,1) \\ \mathbf{r}(1,0) & \mathbf{r}(1,1) & \mathbf{r}_v(1,0) & \mathbf{r}_v(1,1) \\ \mathbf{r}_u(0,0) & \mathbf{r}_u(0,1) & \mathbf{r}_{uv}(0,0) & \mathbf{r}_{uv}(0,1) \\ \mathbf{r}_u(1,0) & \mathbf{r}_u(1,1) & \mathbf{r}_{uv}(1,0) & \mathbf{r}_{uv}(1,1) \end{bmatrix} \begin{bmatrix} \alpha_0(v) \\ \alpha_1(v) \\ \beta_0(v) \\ \beta_1(v) \end{bmatrix}$$

where a point on this patch is located at $(\xi+u, \eta+v)$, with $0 \leq u \leq 1$ and $0 \leq v \leq 1$. In terms of u and v , the patch corners thus are



This can be written more concisely as

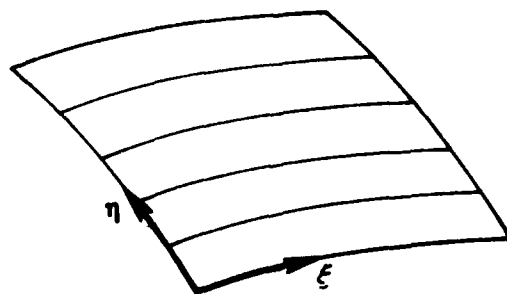
$$\underline{r}(u,v) = F(u) Q F^T(v) \quad (2)$$

with the obvious definitions of F and Q .

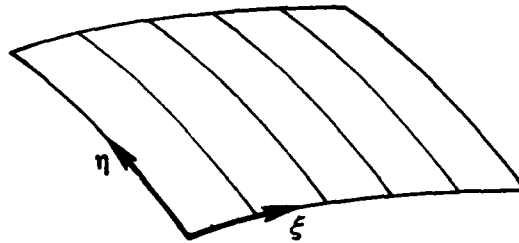
For cubic splines, the blending functions in F are

$$F = \begin{bmatrix} \alpha_0(u) \\ \alpha_1(u) \\ \beta_0(u) \\ \beta_1(u) \end{bmatrix} = \begin{bmatrix} 1 - 3u^2 + 2u^3 \\ 3u^2 - 2u^3 \\ u - 2u^2 + u^3 \\ -u^2 + u^3 \end{bmatrix} \quad (3)$$

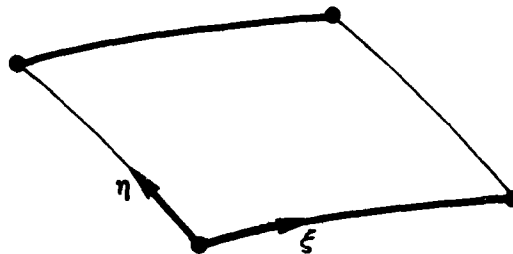
The elements of Q are determined by splining the surface as follows. First, each set of points in the ξ -direction is splined (using quadratic ends, i.e., constant curvature at the ends):



This determines r_{ξ} at each grid point. Independently, each set of points in the η -direction is splined:

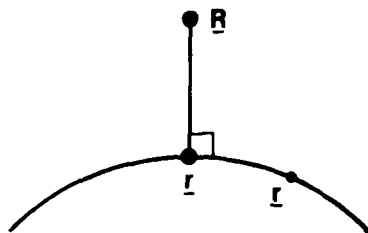


determining r_{η} at each grid point. Next the values of r_{ξ} on the two edges on which η is constant are splined using specified zero values of $(r_{\xi})_{\eta} = r_{\xi\eta}$ at the ends of these curves, i.e., at the corners of the surface:



This gives $(r_{\xi})_{\eta} = r_{\xi\eta}$ at each point on these two edges. Finally, the values of r_{η} for each set of points in the ξ -direction are splined using the specified values of $(r_{\eta})_{\xi} = r_{\xi\eta}$ at the ends. This provides $(r_{\eta})_{\xi} = r_{\xi\eta}$ at each grid point, thus completing the determination of the elements of Q .

With Q known, Eq. (2) locates any point at $(\xi+u, \eta+v)$ on the patch. Now the foot of the normal to the surface from an adjacent field point will be at the location of the surface point closest to the field point in question:



With the field point at \underline{R} , the square of the distance to a point \underline{r} on the surface is $(\underline{R}-\underline{r}) \cdot (\underline{R}-\underline{r})$ so that the minimum distance is determined by the system

$$(\underline{R}-\underline{r}) \cdot \underline{r}_{\xi} = 0$$

$$(\underline{R}-\underline{r}) \cdot \underline{r}_{\eta} = 0$$

On the surface spline we have $\underline{r}_{\xi} = \underline{r}_u$ and $\underline{r}_{\eta} = \underline{r}_v$, and in order to avoid scaling problems the system used in the code is

$$FU(u, v) = (\underline{R} - \underline{r}) \cdot \frac{\underline{r}_u}{|\underline{r}_u|} = 0 \quad (5a)$$

$$FV(u, v) = (\underline{R} - \underline{r}) \cdot \frac{\underline{r}_v}{|\underline{r}_v|} = 0 \quad (5b)$$

which is solved by Newton iteration.

From Eq. (2),

$$\begin{aligned} r_u &= F'(u) Q F^T(v) \\ r_v &= F(u) Q F'^T(v) \end{aligned} \quad (6)$$

and

$$F'(u) = \begin{bmatrix} 1 - 6u + 6u^2 \\ 6u - 6u^2 \\ 1 - 4u + 3u^2 \\ -2u + 3u^2 \end{bmatrix} \quad (7)$$

The iteration then is

$$\begin{bmatrix} u \\ v \end{bmatrix}^{n+1} = \begin{bmatrix} u \\ v \end{bmatrix}^n + \Delta \begin{bmatrix} u \\ v \end{bmatrix} \quad (8)$$

with

$$J \Delta \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} FU \\ FV \end{bmatrix} \quad (9)$$

where

$$J = \begin{bmatrix} \frac{\partial FU}{\partial u} & \frac{\partial FU}{\partial v} \\ \frac{\partial FV}{\partial u} & \frac{\partial FV}{\partial v} \end{bmatrix} \quad (10)$$

From (5),

$$J_{11} = \frac{\partial FU}{\partial u} = -|r_u| + \frac{R - r}{|r_u|^3} \cdot [|r_u|^2 r_{uu} - r_u(r_u \cdot r_{uu})]$$

$$J_{12} = \frac{\partial FU}{\partial v} = - \frac{r_u \cdot r_v}{|r_u|} + \frac{R - r}{|r_u|^3} \cdot [|r_u|^2 r_{uv} - r_u(r_u \cdot r_{uv})]$$

$$J_{21} = \frac{\partial FV}{\partial u} = - \frac{r_v \cdot r_v}{|r_v|} + \frac{R - r}{|r_v|^3} \cdot [|r_v|^2 r_{uv} - r_v(r_v \cdot r_{uv})]$$

$$J_{22} = \frac{\partial FV}{\partial v} = - |r_v| + \frac{R - r}{|r_v|^3} \cdot [|r_v|^2 r_{vv} - r_v(r_v \cdot r_{vv})]$$

and from (6)

$$r_{uu} = F''(u) Q F^T(v)$$

$$r_{vv} = F(u) Q F''^T(v) \quad (11)$$

$$r_{uv} = r_{vu} = F'(u) Q F'^T(v)$$

with

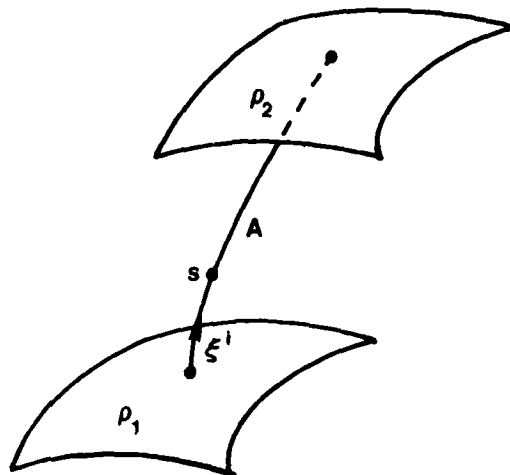
$$F''(u) = \begin{bmatrix} -6 + 12u \\ 6 - 12u \\ -4 + 6u \\ -2 + 6u \end{bmatrix} \quad (12)$$

APPENDIX G

INTERPOLATION FOR RADIUS OF CURVATURE

The radius of curvature in the expressions for control functions given in Appendix C requires some care in order to ensure generality. The construction of the control functions from arc length and curvature contributions developed in that appendix arises from consideration of concentric spheres. Therefore, the baseline distribution for the radius of curvature is linear. However, in general configurations, the two surfaces between which the radius is interpolated may have greatly differing curvatures which may even be of opposite signs. Flat surfaces have infinite radius of curvature, and such surfaces will occur in the field between two surfaces with curvature of opposite signs, as well as on boundaries. Therefore simple linear interpolation for the radius is out of the question in general, but must still be approached in many cases. The construction used is as follows:

The interpolation is between the two ends of a curvilinear coordinate line between two surfaces:

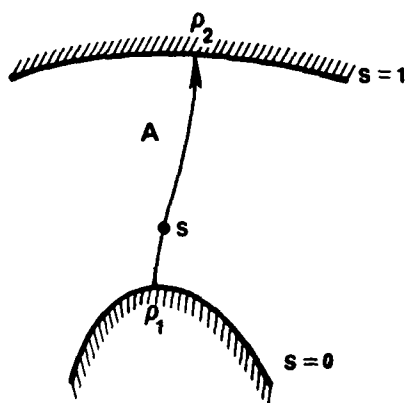


This interpolation is based on the hyperbolic sine function and depends on the sign of the curvature on the two surfaces.

In the following discussion, the radius of curvature of the surface at the lower value of ξ^1 is ρ_1 , and that on the other surface is ρ_2 . The arc length of the curvilinear coordinate line between the surfaces is A , and the relative arc length (range 0-1) at the point on this line where the radius of curvature is interpolated as s .

The interpolation is constructed as a departure from the linear distribution that would prevail between two spherical surfaces. The departure from linearity is made to be slight for some distance from the surface with the smaller radius in order to allow for flat surfaces. (Surfaces with radius of curvature greater than a length scale RSCAL (Section II-C13) are considered to be flat and are given a radius equal to that value.)

When both surfaces have positive curvature



the construction is as follows: If ρ_1 is the smaller, and ρ_2 is equal to the value that would occur for two spherical surfaces separated by a

distance equal to the arc length A , of the connecting coordinate line, then the distribution is linear:

$$\rho(s) = \rho_1 + As$$

If ρ_2 exceeds $\rho_1 + A$, the distribution is

$$\rho(s) = (\rho_1 + As) + f(s)$$

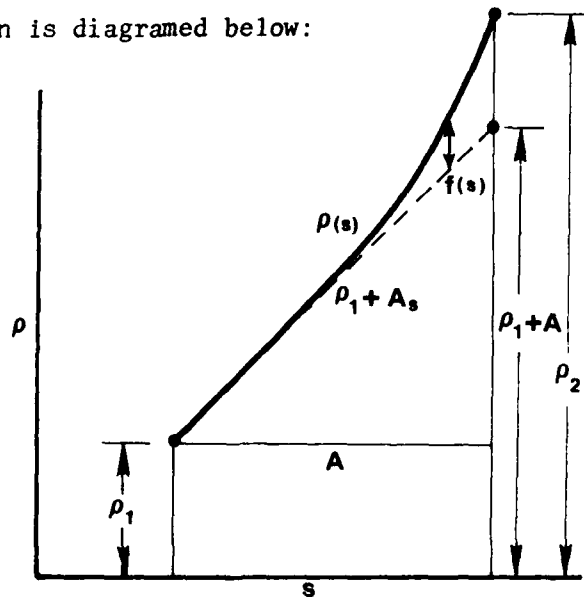
where $f(0) = 0$, $f(1) = \rho_2 - (\rho_1 + A)$, and $f_s(0) = 0$. The function $f(s)$ is made to vary very slowly away from its initial value of zero by using the hyperbolic sine:

$$f(s) = A \left(\frac{\sinh \delta s}{\delta} - s \right)$$

where the parameter δ is determined by the condition on $f(1)$:

$$\frac{\sinh \delta}{\delta} = \frac{\rho_2 - \rho_1}{A}$$

This construction is diagramed below:



Substituting for $f(s)$ in $\rho(s)$, the distribution for this case is then

given by

$$\rho(s) = \rho_1 + (\rho_2 - \rho_1) \frac{\sinh \delta s}{\sinh \delta}$$

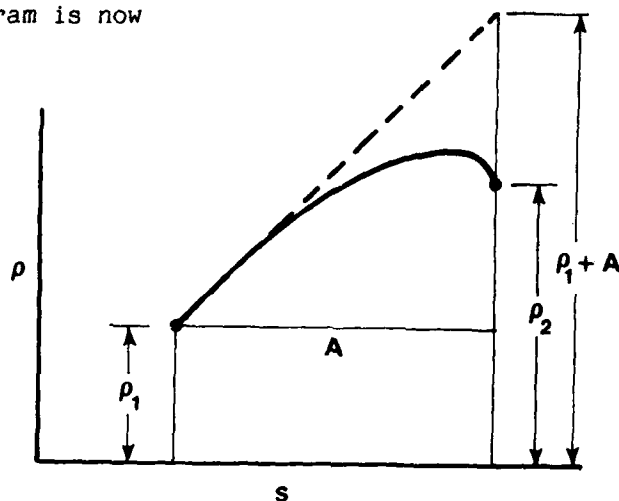
If, however, ρ_2 is less than $\rho_1 + A$, the construction is similar, but with

$$\rho(s) = (\rho_1 + As) - f(s)$$

and now $f(1) = (\rho_1 + As) - \rho_2$, so that δ is determined by

$$\frac{\sinh \delta}{\delta} = \frac{\rho_1 - \rho_2}{A} + 2$$

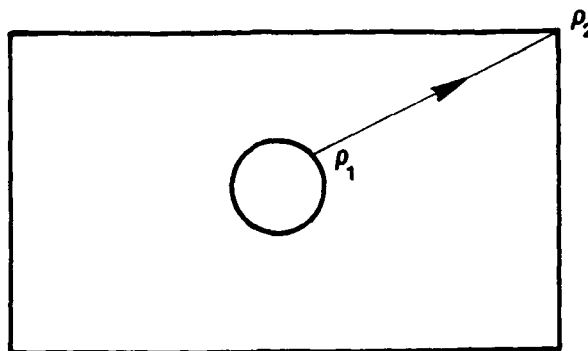
The diagram is now



Substitution for $f(s)$ yields the distribution

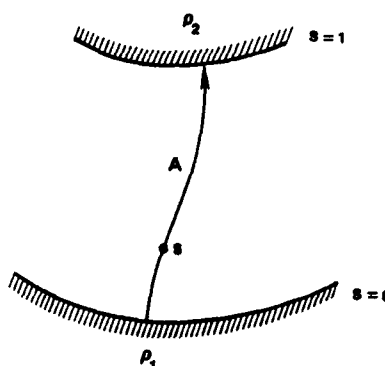
$$\rho(s) = \rho_1 + 2As - (\rho_1 - \rho_2 + 2A) \frac{\sinh \delta s}{\sinh \delta}$$

Note that this form applies even for $\rho_2 < \rho_1$, which occurs, for example, for a circle inside a box:



where the radius of curvature at the corner is much less than that on the circle.

If both surfaces have negative curvature,



the construction is the same, but with ρ_1 in the above replaced by $|\rho_2|$, ρ_2 replaced by $|\rho_1|$, and with s replaced by $1-s$. Thus, with $|\rho_1| > |\rho_2| + A$, we have

$$\rho(s) = - \left[|\rho_2| + (|\rho_1| - |\rho_2|) \frac{\sinh \delta(1-s)}{\sinh \delta} \right]$$

with δ determined by

$$\frac{\sinh \delta}{\delta} = \frac{|\rho_1| - |\rho_2|}{A}$$

while with $|\rho_1| < |\rho_2| + A$, the distribution is

$$\rho(s) = -[|\rho_2| + 2A(1-s) - (|\rho_2| - |\rho_1| + 2A) \frac{\sinh \delta(1-s)}{\sinh \delta}]$$

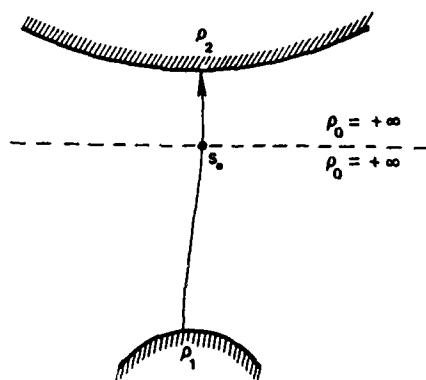
with δ from

$$\frac{\sinh \delta}{\delta} = \frac{|\rho_2| - |\rho_1|}{A} + 2$$

The cases with different curvature on the two surfaces require that the radius of curvature pass through infinity, i.e., that there be a flat surface somewhere in between. With $\rho_1 > 0$ and $\rho_2 < 0$, the location of this flat surface is calculated as

$$s_0 = 1 - \frac{1}{2} \left| \frac{\rho_1}{\rho_2} \right|$$

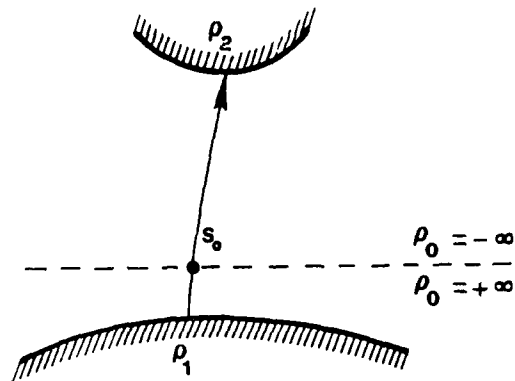
if $\rho_1 < |\rho_2|$:



and by

$$s_0 = \frac{1}{2} \left| \frac{\rho_2}{\rho_1} \right|$$

if $\rho_1 > |\rho_2|$:

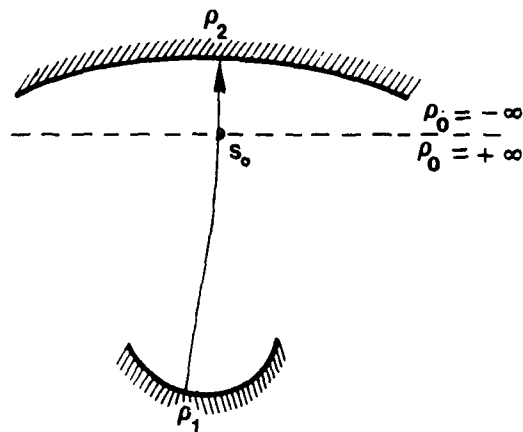


The distribution is then constructed as above, but now separately between ρ_1 and ρ_0 and between ρ_0 and ρ_2 using \pm RSCAL for $\pm\infty$.

If $\rho_1 < 0$ and $\rho_2 > 0$, the flat surface location is calculated from

$$s_0 = 1 - \frac{1}{2} \left| \frac{\rho_1}{\rho_2} \right|$$

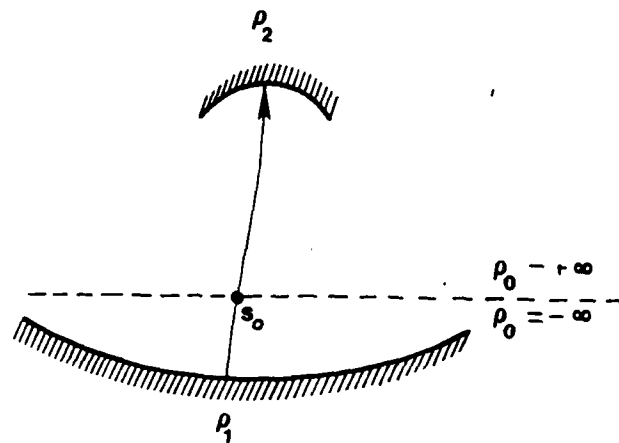
if $\rho_2 > |\rho_1|$



or from

$$s_0 = \frac{1}{2} \left| \frac{\rho_2}{\rho_1} \right|$$

if $\rho_2 < |\rho_1|$:



Again the distribution is constructed separately between ρ_1 and ρ_0 , and between ρ_0 and ρ_2 .

APPENDIX H

FEEDBACK LIMITATION ON ACCELERATION PARAMETERS

Since the iterative adjustment of the control functions for boundary orthogonality is a feedback loop, the SOR acceleration parameters must be limited for stability. This arises as follows.

From the generation equation, Eq. (B-11) of Appendix B, since the central terms of the second derivatives are factored to the left for the iterative solution, the change in r due to a change in the control function is approximately,

$$\Delta r = \omega \frac{\sum_k g^{kk} \Delta P_k r_{\xi k}}{2 \sum_k g^{kk}}$$

where ω is the acceleration parameter. Then

$$r_{\xi i} \cdot \Delta r = \frac{\sum_k g^{kk} g_{ik} \Delta P_k}{2 \sum_k g^{kk}} \quad i=1,2,3$$

Also, from Appendix D, the change in the control functions due to the change in $r_{\xi \xi}$ that occurs when orthogonality is invoked is

$$\Delta P_{\xi} = \frac{n}{\sqrt{g_{\xi \xi}}} \cdot \Delta r = \frac{r_{\xi \xi}}{g_{\xi \xi}} \cdot (2\Delta r)$$

$$\Delta P_m = \frac{1}{g_{\xi \xi}} \left(r_{\xi m} - \frac{g_{m\xi}}{g_{\xi \xi}} r_{\xi \xi} \right) \cdot (2\Delta r)$$

$$\Delta P_n = \frac{1}{g_{\xi \xi}} \left(r_{\xi n} - \frac{g_{n\xi}}{g_{\xi \xi}} r_{\xi \xi} \right) \cdot (2\Delta r)$$

where \underline{n} is the unit normal.

Now, since $g_{lm} = g_{ln} = 0$, we have

$$\begin{pmatrix} \underline{r}_{\xi^l} \cdot \Delta \underline{r} \\ \underline{r}_{\xi^m} \cdot \Delta \underline{r} \\ \underline{r}_{\xi^n} \cdot \Delta \underline{r} \end{pmatrix} = \frac{\omega}{2 \sum_k g^{kk}} \begin{pmatrix} g^{ll} g_{ll} \Delta P_l \\ g^{mm} g_{mm} \Delta P_m + g^{nn} g_{nn} \Delta P_n \\ g^{mm} g_{mn} \Delta P_m + g^{nn} g_{nn} \Delta P_n \end{pmatrix}$$

$$= \frac{\omega}{2 \sum_k g^{kk}} \begin{pmatrix} \Delta P_l \\ \frac{g_{nn} g_{mm}}{G} \Delta P_m + \frac{g_{mm} g_{mn}}{G} \Delta P_n \\ \frac{g_{nn} g_{mn}}{G} \Delta P_m + \frac{g_{mm} g_{nn}}{G} \Delta P_n \end{pmatrix}$$

with $G = g_{mm} g_{nn} - g_{mn}^2$. Substitution for the changes in the control functions then yields, after some cancellations,

$$\begin{pmatrix} \underline{r}_{\xi^l} \cdot \Delta \underline{r} \\ \underline{r}_{\xi^m} \cdot \Delta \underline{r} \\ \underline{r}_{\xi^n} \cdot \Delta \underline{r} \end{pmatrix} = \frac{\omega g^{ll}}{\sum_k g^{kk}} \begin{pmatrix} \underline{r}_{\xi^l} \cdot \Delta \underline{r} \\ \underline{r}_{\xi^m} \cdot \Delta \underline{r} \\ \underline{r}_{\xi^n} \cdot \Delta \underline{r} \end{pmatrix}$$

so that, the stability requirement is

$$\omega < \frac{\sum_k g^{kk}}{g^{ll}}$$

APPENDIX I

PREVENTION OF ONE-DIMENSIONAL OVERLAP

Consider the 1D form of the elliptic generation system, Eq. B-11:

$$x_{\xi\xi} + Px_{\xi} = 0$$

With the switching coefficient A for the first derivative (cf. Section II-F19, p. 167) and a central difference for the second derivative, this becomes

$$(x_{i+1} - 2x_i + x_{i-1}) + P \left(\frac{x_{i+1} - x_{i-1}}{2} \right) + A|P| \left(\frac{x_{i+1} + x_{i-1}}{2} - x_i \right) = 0$$

(Here A=0 gives a central difference for the first derivative and A=1 gives a one-sided difference.) Thus

$$x_i = \frac{(1 + A \frac{|P|}{2})(x_{i+1} + x_{i-1}) + \frac{P}{2}(x_{i+1} - x_{i-1})}{2 + A|P|}$$

Then $x_i \leq x_{i+1}$ requires $A \geq P-2/|P|$ while $x_i \geq x_{i-1}$ requires either $P \geq 0$, or $A \geq 1 - 2/|P|$ if $P < 0$. Thus in general $x_{i-1} \leq x_i \leq x_{i+1}$ if

$$A \geq 1 - \frac{2}{|P|}$$

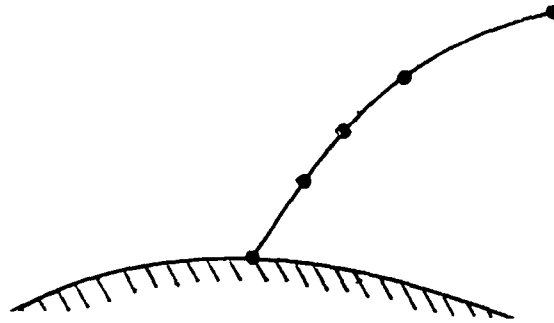
Thus A is taken as

$$A = \begin{cases} 0 & |P| \leq 2 \\ 1 - \frac{2}{|P|} & |P| > 2 \end{cases}$$

APPENDIX J

ZERO-CURVATURE INTERSECTION ON SPLINED SURFACE

The use of zero-curvature Neumann boundary conditions requires that a boundary point moves over the surface to lie at the intersection of the extrapolated grid line and the surface, the extrapolation being done by a straight line extension from the first two points off the surface:



The boundary point is thus determined as the intersection of a straight line and a splined surface. The surface spline is set up as in Appendix F, and the intersection is determined by Newton iteration, also as in that appendix but with a different nonlinear function.

Since a point on the surface spline is given by Eq. (F-2), the intersection is determined by the solution of

$$\underline{r}(u,v) = \underline{R} + \underline{t}s \quad (1)$$

where \underline{t} is the unit vector from \underline{R}_1 to \underline{R} , i.e., tangent to the incoming grid line at \underline{R} . The Newton iteration thus solves for the solution (u,v,s) of

$$\underline{F}(u,v,s) = (\underline{R} - \underline{r}) + \underline{t}s = 0 \quad (2)$$

The iteration then is

$$J\Delta \begin{pmatrix} u \\ v \\ s \end{pmatrix} = - \begin{pmatrix} F^1 \\ F^2 \\ F^3 \end{pmatrix} \quad (3)$$

where

$$J = \begin{pmatrix} F_u^1 & F_v^1 & F_s^1 \\ F_u^2 & F_v^2 & F_s^2 \\ F_u^3 & F_v^3 & F_s^3 \end{pmatrix} \quad (4)$$

From (2),

$$\underline{F}_u = - \underline{r}_u \quad (5a)$$

$$\underline{F}_v = - \underline{r}_v \quad (5b)$$

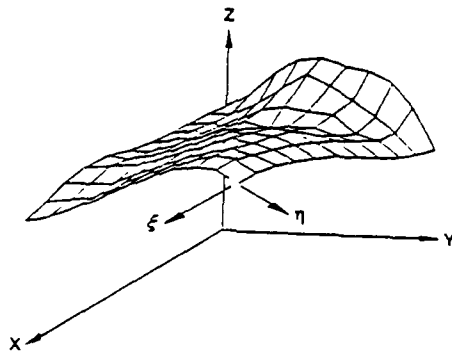
$$\underline{F}_s = \underline{t} \quad (5c)$$

where \underline{r}_u and \underline{r}_v given by Eq. (F-6). (The \underline{F} of this appendix should not be confused with the F of Appendix F.)

Appendix K

SURFACE GRID GENERATION SYSTEMS

The grid generation system given by Eq. (11) of Appendix B (Warsi [1]) is for generation in general three-dimensional regions. The two-dimensional form of this system serves to generate a grid in general two-dimensional regions in a plane. It is also of interest, however, to generate two-dimensional grids on general curved surfaces:



Here the surface is specified, and the problem is to generate a two-dimensional grid on that surface, the third curvilinear coordinate being constant on the surface. The configurations of the transformed region will be the same as for the general case, i.e., composed of contiguous rectangular blocks in a plane, with point locations and/or coordinate line slopes specified on the boundaries. These boundaries now correspond to curves on the curved surface of the physical region. The problem is thus essentially the same as that discussed for two-dimensional plane regions, except that the curvature of the surface must now enter the partial differential equations which comprise the grid generation system.

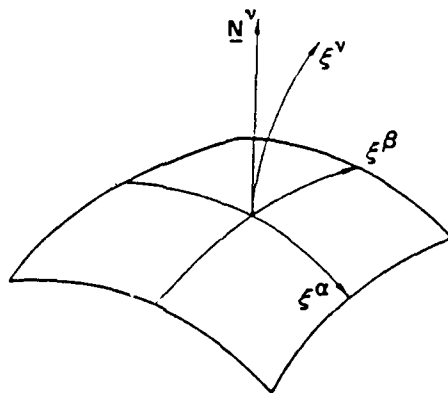
As for general regions, algebraic generation systems based on interpolation can be constructed. The problem can also be considered as an elliptic boundary-value problem on the surface with the same general features discussed in Appendix B being exhibited by the elliptic generation system.

Surface grid generation (Warsi [1])

An elliptic generation system for surface grids can be devised from the formulae of Gauss and Beltrami. The starting point is the set formed by the formulae of Gauss for a surface, which for a surface, $\xi^v = \text{constant}$, ($v = 1, 2$, or 3) are given by

$$r_{\xi^{\alpha}\xi^{\beta}} = \sum_{\delta} T_{\alpha\beta}^{\delta} r_{\xi^{\delta}} + b_{\alpha\beta} n^{(v)} \quad (1)$$

where the variation of the indices α, β and δ is over the two coordinate indices different from v . (Greek coordinate indices are used here to set apart the coordinates generated on a surface from those generated in a three-dimensional region in general).



The unit normal $\underline{n}^{(v)}$, the coefficients $b_{\alpha\beta}$, and the surface Christoffels (T), are given by Eqs. (15), (20), (34), (35) of Appendix A of Ref. [1]. The indices α, β each assume the two values different from v . For each v , with (α, β, v) taken in cyclic order, we have

$$\underline{r}_{\xi\alpha\xi\alpha} = \sum_{\delta} T_{\alpha\alpha}^{\delta} \underline{r}_{\xi\delta} + \underline{n}^{(v)} b_{\alpha\alpha} \quad (2a)$$

$$\underline{r}_{\xi\alpha\xi\beta} = \sum_{\delta} T_{\alpha\beta}^{\delta} \underline{r}_{\xi\delta} + \underline{n}^{(v)} b_{\alpha\beta} \quad (2b)$$

$$\underline{r}_{\xi\beta\xi\beta} = \sum_{\delta} T_{\beta\beta}^{\delta} \underline{r}_{\xi\delta} + \underline{n}^{(v)} b_{\beta\beta} \quad (2c)$$

with δ assuming the two values, α, β .

A surface grid generation system that is analogous in form to that based on the Poisson-type equations in a plane given in Appendix B can be constructed by multiplying Eq. (2a,b,c), respectively, by $G_v g^{\alpha\alpha}$, $2G_v g^{\alpha\beta}$, $G_v g^{\beta\beta}$ and adding. This gives, after some algebra,

$$L^{(v)} \underline{r} + G_v [(\Delta_2^{(v)} \xi^{\alpha}) \underline{r}_{\xi\alpha} + (\Delta_2^{(v)} \xi^{\beta}) \underline{r}_{\xi\beta}] = \underline{n}^{(v)} R^{(v)} \quad (3)$$

where

$$G_v = g_{\alpha\alpha} g_{\beta\beta} - g_{\alpha\beta}^2 = g g^{vv} \quad (4)$$

$$L^{(v)} = g_{\beta\beta} \frac{\partial^2}{\partial \xi^{\alpha} \partial \xi^{\alpha}} - 2g_{\alpha\beta} \frac{\partial^2}{\partial \xi^{\alpha} \partial \xi^{\beta}} + g_{\alpha\alpha} \frac{\partial^2}{\partial \xi^{\beta} \partial \xi^{\beta}} \quad (5)$$

$$R^{(v)} = G_v (k_I^{(v)} + k_{II}^{(v)}) \quad (6)$$

$$k_I^{(v)} + k_{II}^{(v)} = g^{\alpha\alpha} b_{\alpha\alpha} + 2g^{\alpha\beta} b_{\alpha\beta} + g^{\beta\beta} b_{\beta\beta} \quad (7)$$

The quantities $k_I^{(v)}$ and $k_{II}^{(v)}$ are the local principal curvatures of the surface $\xi^v = \text{constant}$. It must be noted here the $R^{(v)}$ as defined in (6) is based on the intrinsic values of $b_{\alpha\beta}$. That is, the $b_{\alpha\beta}$ are solely determined by the data and coordinates as available in the surface.

The operator Δ_2 is called the Beltrami second-order differential operator, and in general is defined as

$$\begin{aligned} \Delta_2^{(v)} = & \frac{1}{\sqrt{G_v}} \left[\frac{\partial}{\partial \xi^\alpha} \left\{ \frac{\partial}{\sqrt{G_v}} (g_{\beta\beta} \frac{\partial}{\partial \xi^\alpha} - g_{\alpha\beta} \frac{\partial}{\partial \xi^\beta}) \right\} \right. \\ & \left. + \frac{\partial}{\partial \xi^\beta} \left\{ \frac{1}{\sqrt{G_v}} (g_{\alpha\alpha} \frac{\partial}{\partial \xi^\beta} - g_{\alpha\beta} \frac{\partial}{\partial \xi^\alpha}) \right\} \right] \end{aligned} \quad (8)$$

Thus

$$\Delta_2^{(v)} \xi^\alpha = \frac{1}{\sqrt{G_v}} \left[\frac{\partial}{\partial \xi^\alpha} \left(\frac{g_{\beta\beta}}{\sqrt{G_v}} \right) - \frac{\partial}{\partial \xi^\beta} \left(\frac{g_{\alpha\beta}}{\sqrt{G_v}} \right) \right] \quad (9a)$$

$$\Delta_2^{(v)} \xi^\beta = \frac{1}{\sqrt{G_v}} \left[\frac{\partial}{\partial \xi^\beta} \left(\frac{g_{\alpha\alpha}}{\sqrt{G_v}} \right) - \frac{\partial}{\partial \xi^\alpha} \left(\frac{g_{\alpha\beta}}{\sqrt{G_v}} \right) \right] \quad (9b)$$

The generation system is now formed by taking, in analogy with Eq. (7) of Appendix B,

$$\Delta_2^{(v)} \xi^\alpha = \sum_{\mu} \sum_{\sigma} g^{\mu\sigma} p_{\mu\sigma}^\alpha \quad (10a)$$

$$\Delta_2^{(v)} \xi^\beta = \sum_{\mu} \sum_{\sigma} g^{\mu\sigma} p_{\mu\sigma}^\beta \quad (10b)$$

where μ and σ each assume the two values α, β in the summation. Here the $P_{\mu\sigma}^\delta$ are the symmetric control functions. Thus the equations for the generation of surface grids are (with $\underline{r} = ix + jy + kz$)

$$D^{(\nu)} \underline{r} = \underline{n}^{(\nu)} R^{(\nu)} \quad (11)$$

where

$$\begin{aligned} D^{(\nu)} = & g_{\beta\beta} \frac{\partial^2}{\partial \xi^\alpha \partial \xi^\alpha} - 2g_{\alpha\beta} \frac{\partial^2}{\partial \xi^\alpha \partial \xi^\beta} + g_{\alpha\alpha} \frac{\partial^2}{\partial \xi^\beta \partial \xi^\beta} \\ & + S \frac{\partial}{\partial \xi^\alpha} + T \frac{\partial}{\partial \xi^\beta} \end{aligned} \quad (12)$$

$$S = g_{\beta\beta} P_{\alpha\alpha}^\alpha - 2g_{\alpha\beta} P_{\alpha\beta}^\alpha + g_{\alpha\alpha} P_{\beta\beta}^\alpha \quad (13)$$

$$T = g_{\beta\beta} P_{\alpha\alpha}^\beta - 2g_{\alpha\beta} P_{\alpha\beta}^\beta + g_{\alpha\alpha} P_{\beta\beta}^\beta \quad (14)$$

The left-hand side of Eq. (11) here corresponds exactly to the 2D form of Eq. (11) of Appendix B, for the plane. However, here we have the complete 3D forms of the metric elements.

$$g_{\alpha\alpha} = x_\xi^\alpha^2 + y_\xi^\alpha^2 + z_\xi^\alpha^2 \quad (15a)$$

$$g_{\beta\beta} = x_\xi^\beta^2 + y_\xi^\beta^2 + z_\xi^\beta^2 \quad (15b)$$

$$g_{\alpha\beta} = x_\xi^\alpha x_\xi^\beta + y_\xi^\alpha y_\xi^\beta + z_\xi^\alpha z_\xi^\beta \quad (15c)$$

The effect of the surface curvature enters through the inhomogenous term, in particular through $R^{(\nu)}$, which is, in fact, equal to twice the product of $\sqrt{G_\nu}$ and the mean curvature of the surface. Here, as for the

plane, the control functions, $p_{\alpha\beta}^\delta$, are considered to be specified. This system corresponds to the following system in the physical space, from (10),

$$\Delta_2^{(v)} \xi^\alpha = \frac{S}{G_v} \quad (16a)$$

$$\Delta_2^{(v)} \xi^\beta = \frac{T}{G_v} \quad (16b)$$

Thus the Beltrami operator on the general surface replaces the Laplacian operator in the plane. If the surface is a plane, the Beltrami operator reduces to the Laplacian.

If only the two control functions $p_{\alpha\alpha}^\alpha$ and $p_{\beta\beta}^\beta$ are included, the surface grid generation reduces to the more practical system

$$g_{\beta\beta} (r_{\xi^\alpha \xi^\alpha} + P r_{\xi^\alpha}) + g_{\alpha\alpha} (r_{\xi^\beta \xi^\beta} + Q r_{\xi^\beta}) - 2g_{\alpha\beta} r_{\xi^\alpha \xi^\beta} = n^{(v)} R^{(v)} \quad (17)$$

corresponding to the plane system given by the 2D form, Eq. (10) of Chapter 4. In the physical plane this system is

$$\Delta_2^{(v)} \xi^\alpha = \frac{g_{\beta\beta}}{G_v} P \quad (18a)$$

$$\Delta_2^{(v)} \xi^\beta = \frac{g_{\alpha\alpha}}{G_v} Q \quad (18b)$$

Equation (3) is the basic equation for the generation of curvilinear coordinates in a given surface. From (6) the function $R^{(v)}$ depends on the principal curvatures $k_I^{(v)}$ and $k_{II}^{(v)}$. The sum $k_I^{(v)} + k_{II}^{(v)}$ is twice the mean curvature of the surface, and its value is invariant to the

coordinates introduced in the given surface. If the equation of the surface in the form $x_3 = f(x_1, x_2)$ is available, then from elementary differential geometry

$$k_I^{(v)} + k_{II}^{(v)} = [(1 + q^2)r - 2pqs + (1 + p^2)t]/(1 + p^2 + q^2)^{3/2} \quad (19)$$

where

$$p = f_{x_1}, \quad q = f_{x_2}, \quad r = f_{x_1 x_1}, \quad s = f_{x_1 x_2}, \quad t = f_{x_2 x_2}.$$

For arbitrary surfaces it is always possible to use a numerical method, e.g., the least square method, to fit an equation in the form $x_3 = f(x_1, x_2)$ or $F(x_1, x_2, x_3) = 0$ and to obtain the needed partial derivatives to find $k_I^{(v)} + k_{II}^{(v)}$ as a function of x_1, x_2, x_3 .

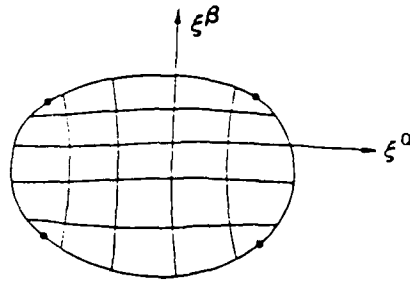
It is more general, however, to generate a new coordinate system based on an already existing coordinate system in a given surface. Let the surface on which the new grid is to be generated be specified parametrically by

$$\underline{r} = \underline{r}(u, v) \quad (20)$$

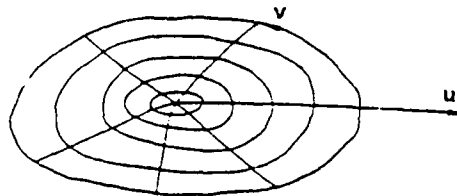
(For example, the parameters (u, v) might be latitude and longitude on a spherical surface.) If the specified Cartesian coordinates on the surface form a finite set of discrete points, a smooth interpolation scheme is needed to recover the differentiable functions in (20). To attain the desired smoothness in the parametric representation (20), it is generally preferable to divide the given surface into a suitable number of patches such that each patch is representable by a bicubic spline with

suitable blending functions. Having once established the smooth parametric functions (20), it is now possible to introduce any other desired coordinate system, say (ξ^α, ξ^β) on the surface.

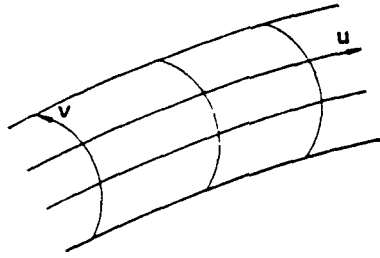
For example, a surface coordinate system ξ^α, ξ^β of the configuration



might be generated on a surface defined by the parametric coordinates (u, v) in a latitude-longitude configuration:



Alternatively, a surface may be defined in terms of cross-sections, in which case one of the parametric coordinates (u, v) runs around the section and the other connects the sections:



The fundamental equations for the generation of (ξ^α, ξ^β) on the surface $\xi^\nu = \text{constant}$ can be obtained from Eq. (17) in the form

$$L^{(\nu)} \underline{r} + g_{\beta\beta} P \underline{r}_{\xi^\alpha} + g_{\alpha\alpha} Q \underline{r}_{\xi^\beta} = \underline{n}^{(\nu)} R^{(\nu)} \quad (21)$$

Here we have taken

$$\Delta_2^{(\nu)} \xi^\alpha = \frac{g_{\beta\beta}}{G_\nu} P \quad (22a)$$

$$\Delta_2^{(\nu)} \xi^\beta = \frac{g_{\alpha\alpha}}{G_\nu} Q \quad (22b)$$

Now using the chain rule of differentiation, we can write $\underline{r}_{\xi^\beta}$, $\underline{r}_{\xi^\alpha}$, $\underline{r}_{\xi^\alpha \xi^\beta}$, etc., in terms of \underline{r}_u , \underline{r}_v , \underline{r}_{uu} , etc. Thus, for example,

$$\underline{r}_{\xi^\alpha} = \underline{r}_u^u \xi^\alpha + \underline{r}_v^v \xi^\alpha \quad (23)$$

$$\begin{aligned} \underline{r}_{\xi^\alpha \xi^\beta} &= \underline{r}_u^u \xi^\alpha \xi^\beta + \underline{r}_v^u \xi^\alpha \xi^\beta + (\underline{r}_{uu}^u \xi^\beta + \underline{r}_{uv}^v \xi^\beta) u \xi^\alpha \\ &\quad + (\underline{r}_{uv}^u \xi^\beta + \underline{r}_{vv}^v \xi^\beta) v \xi^\alpha \end{aligned} \quad (24)$$

$$\begin{aligned} g_{\alpha\beta} &= \underline{r}_{\xi^\alpha} \cdot \underline{r}_{\xi^\beta} = (\underline{r}_u \cdot \underline{r}_u) u \xi^\alpha u \xi^\beta + (\underline{r}_v \cdot \underline{r}_v) v \xi^\alpha v \xi^\beta \\ &\quad + (\underline{r}_u \cdot \underline{r}_v) (u \xi^\alpha v \xi^\beta + u \xi^\beta v \xi^\alpha) \end{aligned}$$

$$g_{\alpha\beta} = \bar{g}_{\alpha\alpha} u_{\xi}^{\alpha} u_{\xi}^{\beta} + \bar{g}_{\beta\beta} v_{\xi}^{\alpha} v_{\xi}^{\beta} + \bar{g}_{\alpha\beta} (u_{\xi}^{\alpha} v_{\xi}^{\beta} + v_{\xi}^{\alpha} u_{\xi}^{\beta}) \quad (25)$$

with the \bar{g} quantities as defined below. Substituting these derivatives in (21), we get

$$\begin{aligned} r_u^{(L^{(v)})u} + g_{\beta\beta} p u_{\xi}^{\alpha} + g_{\alpha\alpha} q u_{\xi}^{\beta} + r_v^{(L^{(v)})v} + g_{\beta\beta} p v_{\xi}^{\alpha} + g_{\alpha\alpha} q v_{\xi}^{\beta} \\ + J_v^2 \bar{L}^{(v)} r = \bar{n}^{(v)} R^{(v)} J_v^2 \end{aligned} \quad (26)$$

where

$$\bar{L}^{(v)} = \bar{g}_{\beta\beta} \frac{\partial^2}{\partial u \partial u} - 2\bar{g}_{\alpha\beta} \frac{\partial^2}{\partial u \partial v} + \bar{g}_{\alpha\alpha} \frac{\partial^2}{\partial v \partial v} \quad (27)$$

$$J_v = u_{\xi}^{\alpha} v_{\xi}^{\beta} - v_{\xi}^{\alpha} u_{\xi}^{\beta} \quad (28)$$

$$\bar{G}_v = \bar{g}_{\beta\beta} \bar{g}_{\alpha\alpha} - \bar{g}_{\alpha\beta}^2 \quad (29)$$

$$\bar{g}_{\beta\beta} = r_v \cdot r_v, \quad \bar{g}_{\alpha\beta} = r_u \cdot r_v, \quad \bar{g}_{\alpha\alpha} = r_u \cdot r_u \quad (30)$$

To isolate the differential equations for u and v as dependent variables from (26), we take the dot product of Eq. (26) with r_u , and then with r_v , and use the conditions

$$r_u \cdot \bar{n}^{(v)} = 0, \quad r_v \cdot \bar{n}^{(v)} = 0$$

Surface grid generation system

The generation system then is

$$a(u_{\xi}^{\alpha} u_{\xi}^{\alpha} + p u_{\xi}^{\alpha}) + c(u_{\xi}^{\beta} u_{\xi}^{\beta} + q u_{\xi}^{\beta}) - 2b u_{\xi}^{\alpha} u_{\xi}^{\beta} = J_v^2 \Delta_2 u \quad (31a)$$

$$a(v_{\xi^{\alpha}\xi^{\alpha}} + P v_{\xi^{\alpha}}) + c(v_{\xi^{\beta}\xi^{\beta}} + Q v_{\xi^{\beta}}) - 2b v_{\xi^{\alpha}\xi^{\beta}} = J_v^2 \Delta_2 v. \quad (31b)$$

where

$$a = \bar{g}_{\beta\beta} v_{\xi^{\beta}}^2 + 2\bar{g}_{\alpha\beta} u_{\xi^{\beta}} v_{\xi^{\alpha}} + \bar{g}_{\alpha\alpha} u_{\xi^{\alpha}}^2 \quad (32a)$$

$$b = \bar{g}_{\beta\beta} v_{\xi^{\beta}} v_{\xi^{\alpha}} + \bar{g}_{\alpha\beta} (u_{\xi^{\beta}} v_{\xi^{\alpha}} + u_{\xi^{\alpha}} v_{\xi^{\beta}}) + \bar{g}_{\alpha\alpha} u_{\xi^{\beta}} u_{\xi^{\alpha}} \quad (32b)$$

$$c = \bar{g}_{\beta\beta} v_{\xi^{\alpha}}^2 + 2\bar{g}_{\alpha\beta} u_{\xi^{\alpha}} v_{\xi^{\alpha}} + \bar{g}_{\alpha\alpha} u_{\xi^{\alpha}}^2 \quad (32c)$$

$$\Delta_2 u = \left[\frac{\partial}{\partial u} \left(\frac{\bar{g}_{\beta\beta}}{J_v} \right) - \frac{\partial}{\partial v} \left(\frac{\bar{g}_{\alpha\beta}}{J_v} \right) \right] \bar{J}_v \quad (33a)$$

$$\Delta_2 v = \left[\frac{\partial}{\partial v} \left(\frac{\bar{g}_{\alpha\alpha}}{J_v} \right) - \frac{\partial}{\partial u} \left(\frac{\bar{g}_{\alpha\beta}}{J_v} \right) \right] \bar{J}_v \quad (33b)$$

(Here the Beltramians have been re-defined to be multiplied by \bar{J}_v^2)
 Note that the metric quantities with an overbar relate to the surface definition in terms of the parametric coordinates and therefore can be calculated directly from the surface specification by Eq. (30).

The Beltramians of Eq. (32) are expanded for implementation to

$$\Delta_2 u = \frac{1}{J_v} \{ \bar{J}_v [(\bar{g}_{\beta\beta})_u - (\bar{g}_{\alpha\beta})_v] - [\bar{g}_{\beta\beta}(\bar{J}_v)_u - \bar{g}_{\alpha\beta}(\bar{J}_v)_v] \} \quad (34a)$$

$$\Delta_2 v = \frac{1}{J_v} \{ \bar{J}_v [(\bar{g}_{\alpha\alpha})_v - (\bar{g}_{\alpha\beta})_u] - [\bar{g}_{\alpha\alpha}(\bar{J}_v)_v - \bar{g}_{\alpha\beta}(\bar{J}_v)_u] \} \quad (34b)$$

with

$$(\bar{g}_{\alpha\alpha})_u = 2r_u \cdot r_{uu} \quad (\bar{g}_{\beta\beta})_u = 2r_v \cdot r_{uv}$$

$$(\bar{g}_{\alpha\alpha})_v = 2r_u \cdot r_{uv} \quad (\bar{g}_{\beta\beta})_v = 2r_v \cdot r_{vv}$$

$$(g_{\alpha\beta})_u = r_v \cdot r_{uu} + r_u \cdot r_{uv}$$

$$(g_{\alpha\beta})_v = r_u \cdot r_{vv} + r_v \cdot r_{uv}$$

$$(\bar{J}_v)_u = - \frac{1}{2\sqrt{G}_v} [\bar{g}_{\alpha\alpha}(\bar{g}_{\beta\beta})_u + \bar{g}_{\beta\beta}(\bar{g}_{\alpha\alpha})_u - 2\bar{g}_{\alpha\beta}(\bar{g}_{\alpha\beta})_u]$$

$$(\bar{J}_v)_v = - \frac{1}{2\sqrt{G}_v} [\bar{g}_{\alpha\alpha}(\bar{g}_{\beta\beta})_v + \bar{g}_{\beta\beta}(\bar{g}_{\alpha\alpha})_v - 2\bar{g}_{\alpha\beta}(\bar{g}_{\alpha\beta})_v]$$

Further discussion is given in Reference 17.

OPERATIONS LIST - INPUT

AVERAGE.	121
BLOCK.	116
CONTROL.	123
CUT.	120
EXTRAP.	125
FIELD.	121
FILE.	118
FIX.	119
IMAGE.	121
INITIAL.	123
INTERP.	121
LIST.	119
NEUMAN.	119
NEUMAN1 or NEUMAN2 or NEUMAN3.	119
ORTHOG.	120
OUT.	119
POINT.	125
REFLECT.	120
SETVAL.	124
SIZE.	124
SMOOTH.	122
SPACE.	123
STORE.	124
SUB.	118
SURFACE.	125
UNFIX.	124

OPERATIONS LIST - OUTPUT

ERROR. 141
INITIAL. 140
PLOT 141
PRINT. 141

SUBROUTINES LIST

ACOSG.	273
AITKEN	272
AVGCON	263
BLKON.	275
BONCON	263
CCDR	269
CCDW	268
CCD1	269
CCD2	269
CCD3	269
CHKCON	274
CHKINT	265
CHKNEU	273
CHKORT	273
CHKREF	274
CHKSET	265
CHKSUB	265
CHKTYP	264
CONFUN	250
CONINT	211
CONLINE.	200
CONLINR.	208
CONSETE.	226
CONSETU.	224
CONSURF.	187
CONSURR.	204
COSG	273
CUTCON	264
DEFAULT	233
DEFCON	264
DERP3.	280
EXTCON	251
EXTCOR	250
FIXIMG	150
IMGIMG	146
IMGPTS	151
JACBCK	250
LIM.	265
MERP2, MERP3	280
MOVEIN	275
NEUIDX	249
NEUPTS	251
OFF.	197
OPTACC	219
ORIDX.	249
PRTGRD	270
READB.	274
READF.	269
READL.	270
READS.	274
REDRES	272

REFCON	263
REFPTS	261
R2DPTS	262
SETACC	242
SETAXS	240
SETIMG	242
SETIMI	244
SETIMO	244
SETIMP	154
SETIMR	152
SETNEU	234
SETNOR	241
SETOBJ	237
SETORT	235
SETREF	236
SETRES	234
SETR2D	240
SETSPA	237
SETSPL	238
SETYPE	234
SING	273
SMOOTH	223
SPLCUR	232
SPLINT	278
SPLSUR	231
SSD	145
SSDR	268
SSDW	267
SSD1	268
SSD3	268
SURCOR	280
SURSYS	275
TANG	273
TERPR	187
TERP1	182
TERP2	184
TERP3	186
TRANS	158
VOLSYS	212
WRTGRD	271
WRTPLT	271
WRTRES	272
WRTRRR	272
WRTXYZ	271

REFERENCES

1. THOMPSON, J. F., WARSI, Z.U.A., and MASTIN, C. W., Numerical Grid Generation: Foundations and Applications, North-Holland, 1985.
2. THOMPSON, J. F., WARSI, Z.U.A., and MASTIN, C. W., "Boundary-Fitted Coordinate Systems for Numerical Solution of Partial Differential Equations - A Review," Journal of Computational Physics, Vol. 47, p. 1, 1982.
3. THOMPSON, J. F., "Grid Generation Techniques in Computational Fluid Dynamics," AIAA Journal, Vol. 22, p. 1505, 1984.
4. THOMPSON, J. F., "A Survey of Dynamically-Adaptive Grids in the Numerical Solution of Partial Differential Equations," Applied Numerical Mathematics, Vol. 1, p. 3, 1985.
5. EISEMAN, P.R., "Grid Generation for Fluid Mechanics Computations," Annual Review of Fluid Mechanics, Vol. 17, 1985.
6. THOMPSON, J. F., (Ed.), Numerical Grid Generation, North-Holland 1982. (Also published as Vol. 10 and 11 of Applied Mathematics and Computation, 1982).
7. SMITH, ROBERT E., (Ed.) Numerical Grid Generation Techniques, NASA Conference Publication 2166, NASA Langley Research Center, 1980.
8. GHIA, K.N., and GHIA, U., (Ed.), Advances in Grid Generation, FED-Vol. 5, ASME Applied Mechanics, Bioengineering, and Fluids Engineering Conference, Houston, 1983.
9. HAUSER J., and TAYLOR, C., (Ed.) Numerical Grid Generation in Computational Fluid Dynamics, Pineridge Press, 1986.
10. THOMPSON, J.F., (ed.), "A Survey of Composite Grid Generation for General Three-Dimensional Regions," in Numerical Methods for Engine-Airframe Interpolation, S.N.B. Murthy and G.C. Paynter, (Ed.), AIAA, 1986.
11. THOMAS, P.D., "Composite Three-Dimensional Grids Generated by Elliptic Systems," AIAA Journal, Vol. 20, p. 1195, 1982.
12. SORENSON, R.L., "Three-Dimensional Elliptic Grid Generation about Fighter Aircraft for Zonal Finite-Difference Computations," AIAA-86-0429, AIAA 24th Aerospace Sciences Conference, Reno, 1986.

13. EHRLICH, LOUIS W., "An Ad Hoc SOR Method," Journal of Computational Physics, Vol. 44, p. 31, 1981.
14. GORDON, W.J., "Blending Function Methods of Bivariate and Multivariate Interpolation," SIAM J. of Numerical Analysis, 8, 158, 1971.
15. GORDON, WILLIAM J. and THEIL, LINDA C., "Transfinite Mappings and Their Application to Grid Generation," Numerical Grid Generation, Joe. F. Thompson, (Ed.), North-Holland, 1982.
16. FAUX, I.D. and PRATT, M.J., Computational Geometry for Design and Manufacture, Ellis Horwood, 1979.
17. Jones, G.A., "Surface Grid Generation for Composite Block Grids," PhD Dissertation, Mississippi State University, May 1988.